60436300

CONTROL DATA CORPORATION

NOS VERSION 1
BATCH USER'S GUIDE

CDC® COMPUTER SYSTEMS:
CYBER 170
MODELS 171, 172, 173, 174, 175
CYBER 70
MODELS 71, 72, 73, 74
6000 SERIES

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(9-1-77)	
Publication No. 60436300	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV
Front cover	-	Cmt sheet	A						
Inside front cover	-	Back cover	-						
Title page	-								
ii	A								
iii/iv	A								
v/vi	A								
vii	A								
viii	A								
1-1	A								
2-1	A								
2-2	A								
2-3	A								
3-1	A								
3-2	A								
3-3	A								
3-4	A								
3-5	A								
3-6	A								
3-7	A								
4-1	A								
4-2	A								
4-3	A								
4-4	A								
4-5	A								
5-1	A								
5-2	A								
5-3	A								
5-4	A								
5-5	A								
5-6	A								
5-7	A								
6-1	A								
6-2	A								
6-3	A								
6-4	A								
6-5	A								
6-6	A								
7-1	A								
7-2	A								
8-1	A								
8-2	A								
8-3	A								
8-4	A								
9-1	A								
9-2	A								
9-3	A								
9-4	A								
9-5	A								
A-1	A								
A-2	A								
B-1	A								
C-1	A								
C-2	A								
D-1	A								
D-2	A								
E-1	A								
Index-1	A								
Index-2	A								

PREFACE

The Network Operating System (NOS) is used with CONTROL DATA® CYBER 170 Models 171, 172, 173, 174, and 175 Computer Systems; CDC® CYBER 70 Models 71, 72, 73, and 74 Computer Systems; and CDC® 6000 Series Computer Systems.

The NOS Batch User's Guide is an introduction to the use of NOS. It is intended for the applications programmer who is already familiar with a problem-oriented language, such as FORTRAN or COBOL, and wants to extend his capabilities by using the job-control statements of a full-scale operating system. NOS supervises job processing in response to over 150 control statements. This guide explains a fundamental subset of about 50 statements. Only those parameters required for preliminary application are included.

The descriptions in this guide are oriented to a data processing environment in which the user submits a job to a data center and has the job processed without concern for the mechanics of processing. Accordingly, system operation and hardware functions are not mentioned unless a control statement requires identification of a software routine or a hardware feature.

DISCLAIMER

This manual describes a subset of the features and parameters documented in the NOS Reference Manual. Control Data Corporation cannot be responsible for the proper functioning of any features or parameters not documented in the reference manual.

RELATED PUBLICATIONS

This guide does not serve as an introduction to the programming languages used in the examples nor does it give the fundamentals of time-sharing operation in the description of batch jobs from a terminal. The languages and time-sharing operation are covered in the following manuals.

Control Data Publication	Publication No.
FORTRAN Extended 4 Reference Manual	60497800
COBOL 4 Reference Manual	60496800
COBOL 5 Reference Manual	60497200
BASIC 3 Reference Manual	19983900
NOS Time-Sharing User's Reference Manual	60435500

To enlarge on the fundamentals learned in this guide, the user should consult the following manuals.

Control Data Publication	Publication No.
NOS Reference Manual, Volume 1	60435400
NOS Export/Import Reference Manual	60436200
NOS Remote Batch Facility Reference Manual	60499600

The preceding manuals contain references to other relevant manuals.

CONTENTS

1. INTRODUCTION 1-1

Motivation for This Guide 1-1
Organization of This Guide 1-1

2. FILES AND RECORDS 2-1

Delimiters 2-1
- End-of-Record (EOR) 2-1
- End-of-File (EOF) 2-2
- End-of-Information (EOI) 2-2

Input and Output Files 2-2
Local Files 2-2
Permanent Files 2-3

3. BATCH JOBS 3-1

Local Batch 3-1
Remote Batch 3-1
Deferred Batch 3-1
Conversational Batch 3-1
Batch Job Structure 3-1
Control Statement Format 3-2
Control Statement Record 3-3
- Job Control Statement 3-3
- USER Control Statement 3-3
- CHARGE Control Statement 3-4
- Control Statement Record Comments 3-4

Programs 3-5

4. LOCAL FILES 4-1

Copy Statements 4-1
- Binary Copy Statements 4-1
- Coded Copy Statements 4-2

VERIFY Statement 4-3
File Positioning Statements 4-4
- Background 4-4
- Forward Positioning Statements 4-4
- Backward Positioning Statements 4-4

5. PERMANENT FILES 5-1

Indirect Access Permanent Files 5-1
- How to Create an Indirect Access Permanent File 5-1
- How to Access an Indirect Access Permanent File 5-2
- How to Add Information to an Indirect Access Permanent File 5-2
- How to Modify an Indirect Access Permanent File 5-2

Direct Access Permanent Files 5-3
- How to Create a Direct Access Permanent File 5-3
- How to Access a Direct Access Permanent File 5-3
- How to Modify a Direct Access Permanent File 5-4

Purging Permanent Files 5-4
Alternate Access of Permanent Files 5-4
How to Obtain a Listing of Permanent Files 5-6

6. CONTROL LANGUAGE 6-1

Format 6-1
Expressions 6-1
File Function 6-1
SET Statement 6-1
DISPLAY Statement 6-2
GOTO Statement 6-2
IF Statement 6-3
CALL Statement 6-3

7. ERROR CONTROL 7-1

8. TAPE FILES 8-1

Definitions 8-1
- Tape Tracks 8-1
- Recording mode 8-1
- Parity 8-1
- Blocks 8-2
- Volume 8-2
- Volume Serial Number 8-2
- Tape Labels 8-2
- Tape Format 8-2

How to Create a Labeled Tape 8-2
How to Access a Labeled Tape 8-3
How to Copy from One Tape to Another 8-3
Tape Error Messages 8-4

9. BATCH INPUT FROM A TERMINAL 9-1

Deferred Batch 9-1
- Reformatting Directives 9-1
- SUBMIT Statement 9-2
- ENQUIRE Statement 9-3
- DAYFILE Statement 9-3
- Listing Batch Output at a Terminal 9-4

Conversational Batch 9-4

APPENDIXES

A Glossary A-1
B Dayfile B-1
C Resources Available to the User C-1
D NOS Standard Character Set D-1
E Operators E-1

INDEX

FIGURES

2-1 EOR Punched Card 2-1
2-2 EOF Punched Card 2-2
2-3 EOI Punched Card 2-2
3-1 Batch Job Representation 3-2
3-2 Comment Demonstration 3-5
3-3 Printed Dayfile after Processing 3-5
4-1 File Creation with Copy Statements 4-2
4-2 Output from Processing a Verify Statement 4-4
4-3 Operation of File Positioning Statements on a Multifile File 4-5
5-1 File Update 5-3
5-2 CATLIST Parameter LO=F Heading 5-7
6-1 GOTO Job Demonstration 6-3
6-2 Resulting Dayfile from GOTO 6-3
6-3 Control Statement Record Expansion with Nested Calls 6-5
6-4 CALL(METRIC) Dayfile 6-6
6-5 CALL(METRIC,S=2LENG) Dayfile 6-6
7-1 EXIT Statement Operation 7-1
8-1 Magnetic Tape Data Layout 8-1
9-1 Reformatting a Submit File 9-2

TABLE

3-1 Common Parameters for Language Processor Call Statements 3-7

An operating system processes user jobs. The processing includes all activities associated with data processing (compiling and executing programs, formatting data, calculating, retrieving stored information, and preserving new information). A user job is a unit of work organized by the user to accomplish specific data processing tasks. Minimally, a job contains validation identification followed by a sequence of control statements that specify data manipulations to be performed by the system. Beyond this minimum, a job can contain programs and input for programs.

MOTIVATION FOR THIS GUIDE

Many jobs contain only programs and program data. Although the user need know only a few control statements to get such jobs processed, a repertoire of control statements is more efficient. Single control statements can manipulate tape, create permanent files, copy data, reformat files, or initiate input/output (I/O) actions. Without control statements, the user must write a program every time he performs one or more of these operations.

ORGANIZATION OF THIS GUIDE

This guide is organized for instruction. It is intended for a user who writes programs in support of daily tasks but whose contact with an operating system has been the few control statements required for job processing. This user, to extend his knowledge in using system control statements, should become familiar with sections 2, 3, and 4. He can then either select sections that serve immediate needs or systematically study the remaining sections.

The fundamental unit of data organization under NOS is a file. A user's interaction with the system is in terms of files.

- The job the user submits for processing is a file.
- Collections of data included with the job are transferred to the system in named packets; each packet is a file.
- Control statements in the job can designate a name as a file; this is an empty file to which other statements or programs can add data.
- Control statements in a job can copy all or part of an existing file; the copy is a file.
- The output from job processing is a file.

A file may be empty or as large as the user's resource limitations permit.

The user can subdivide a file into logical records and he can link several files together into a multifile file. He does this subdividing either by means of parameters he includes in programs or control statements that create the files, or he can insert delimiters in the data he submits with a job (refer to Delimiters).

The user subdivides a file into records when he wants to access these subdivisions without accessing the entire file (for example, a master inventory is divided into parts categories). On the other hand, the user links several files into a multifile file when they fit into a single category he wants to access with a single reference (for example, student rosters from a number of classes are consolidated into one departmental roster).

All files are identified and accessed by a file name that is defined by the system or the user. This guide considers two system files, INPUT and OUTPUT (refer to INPUT and OUTPUT Files). The user names his file in the program or control statement that initiates its creation (refer to sections 3 and 4). A file name must be one to seven alphanumeric characters.

DELIMITERS

The user defines a file by assigning it a name and specifying the end of each record and the end of the file with delimiters. The user inserts delimiters directly or indirectly when he creates a file. He does this directly in the job he submits by inserting special punched cards (if the job is a card deck) or typing in special directives (if the job is entered from a terminal). The special punched cards have three or four numbers multipunched in column 1. The user creates these by depressing the MULT PCH key on the keypunch and, while it is down, entering the numbers. The user inserts delimiters indirectly by specifications he includes with control statements or programs. When the control statement is processed, the system puts the delimiters in the specified locations of the file being created.

When a program is run, the system will place delimiters in files created by the program according to specifications in the program.

The beginning of a file is called the beginning-of-information (BOI). BOI is internally recorded by the system when the file is created. No mark is put in the file by the system or the user; however, the user should be aware of its meaning since several control statement descriptions refer to the BOI.

The user can delimit his files through end-of-record (EOR), end-of-file (EOF), and end-of-information (EOI).

END-OF-RECORD (EOR)

The user inserts an EOR in a sequence of information to specify that all the information between this EOR and the previous terminator or BOI is to be accessed as a single record. For a card deck, an EOR is a card with rows 7, 8, and 9 punched in column 1 (refer to figure 2-1). For a batch job submitted from a terminal with the SUBMIT control statement (refer to section 9), the user inserts an EOR by typing /EOR.

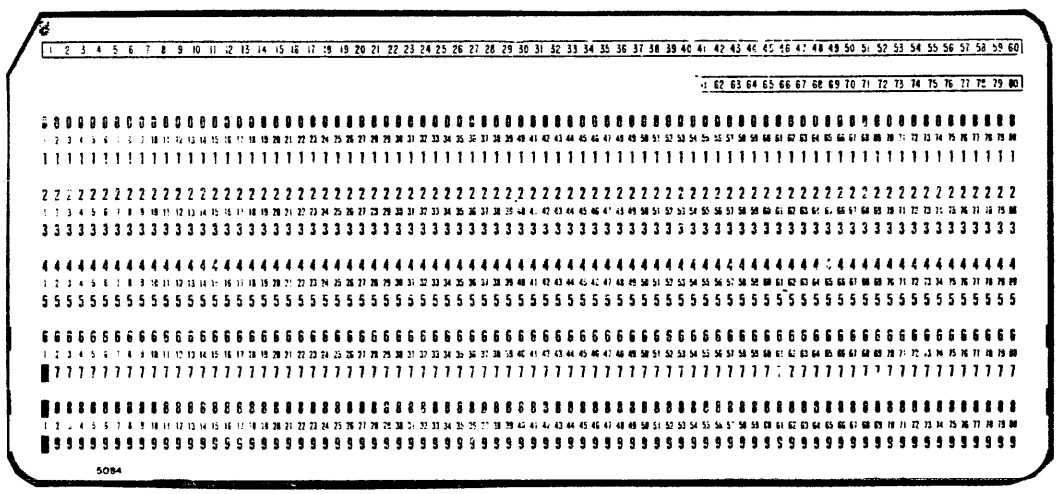


Figure 2-1. EOR Punched Card

END-OF-FILE (EOF)

The user inserts an EOF in a sequence of information to specify that all the information between this EOF and the previous EOF or BOI is to be considered a single file. For a card deck, an EOF is a card with rows 6, 7, and 9 punched in column 1 (refer to figure 2-2). For a batch job submitted from a terminal with the SUBMIT control statement (refer to section 9), the user inserts an EOF by typing /EOF.

END-OF-INFORMATION (EOI)

The user or the system adds an EOI to a sequence of information to specify that the entire sequence between the BOI and this EOI is to be referenced by the name assigned to this file or multifile. For a job punched on cards, the last card in the deck must be an EOI, a card with rows 6, 7, 8, and 9 punched in column 1 (refer to figure 2-3). For a batch job from a time-sharing terminal, the user does not have to designate an EOI.

INPUT AND OUTPUT FILES

The job input file is the system file containing the entire job the user submits for processing. Control statements in the job reference this file with the name INPUT.

The job output file is the system file containing the output resulting from program execution and all the data copied to it by control statements. Control statements in the job reference this file with the name OUTPUT.

The job input file is also known as a job file, and the job output file is also known as a print file or punch file.

LOCAL FILES

A local file has the following characteristics.

- It is created by the job being processed.
- It can be accessed only by that job.
- It is no longer accessible when processing terminates.

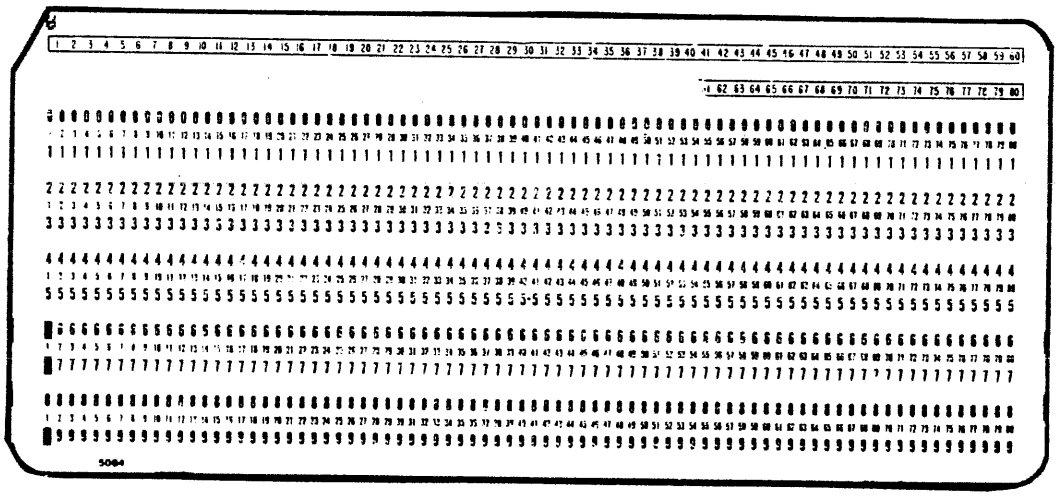


Figure 2-2. EOF Punched Card

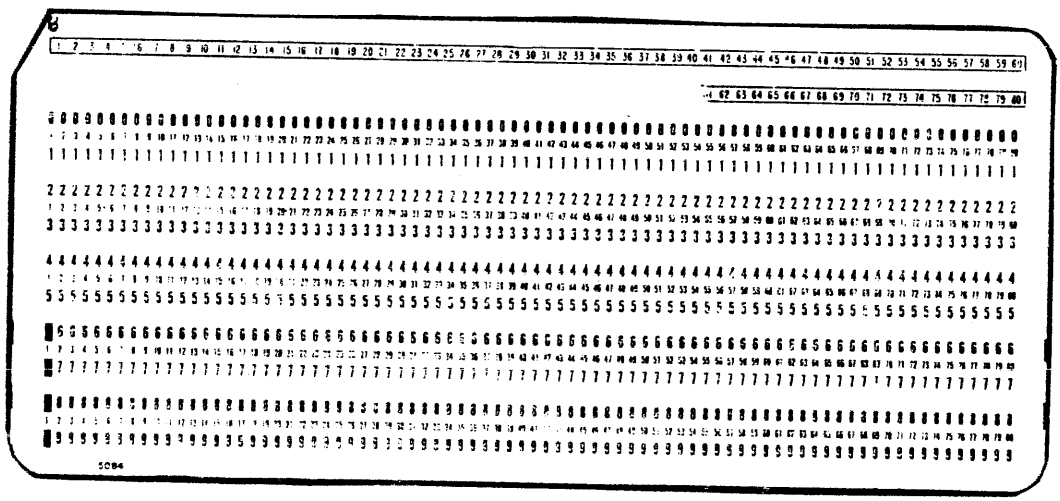


Figure 2-3. EOI Punched Card

Since local files are available only to the job that creates them, they are said to be local to that job (refer to section 3).

PERMANENT FILES

The user can create and access files that remain in the system until they are specifically purged. They are distinct from local files, because their availability is independent of any job processing duration. Two types of permanent files serve two types of needs, indirect access and direct access. Indirect access permanent files typically contain relatively small collections of data, and direct access permanent files typically contain larger collections of data.

The user can create an indirect access permanent file from a local file with special permanent file control statements. This file is accessed by retrieving the permanent file with special permanent file control statements (refer to section 5) to make a local copy of that file.

The user can create a direct access permanent file by reserving an area on mass storage with the DEFINE control statement (refer to section 5) and then writing information in that area. Thereafter, the user can access the file directly with the ATTACH control statement (refer to section 5).

This section describes the types of batch jobs, the structure of a batch job, the format of a control statement, and how programs are added to a job. The following control statements are explained in this section.

job	BASIC
USER	FTN
CHARGE	COBOL
COMMENT	COBOL5

A programmer organizes a job with step-by-step instructions that specify everything to be done in processing the job. These instructions are the first record of the job. If he has programs to compile and execute, he may add these as additional records. If data input is required, he can supply this as additional records in the job or use instructions in the control statement record to reference existing files. Having organized the job, the programmer submits it for processing as a single unit without further intervention on his part. The next thing he sees is the output from processing of the job.

A job can be punched on cards or entered at a terminal. In either case, if the instructions and data are submitted as a complete unit and without further user intervention, they are classed as a batch job. If the instructions are entered one at a time from a terminal with possible system/user interaction after each instruction, they constitute a time-sharing job (conversational batch).

LOCAL BATCH

A local batch job is punched on cards and submitted for processing via a card reader at the local computer site. Output is ordinarily routed to a local printer. Local batch is the assumed situation in the major portion of this guide.

REMOTE BATCH

A remote batch job is submitted for processing via a card reader at a remote batch terminal. Remote implies geographical separation of job submission and job processing (refer to section 9).

The system processes local and remote batch jobs similarly. Output from a remote batch job goes, by default, to the terminal from which it was entered. However, it can be routed to the local site where the computer is located.

DEFERRED BATCH

If a programmer creates a batch job at a time-sharing terminal by entering the statements as they would be punched on cards and then submits this sequence with a single command, the job is called a deferred batch job (refer to section 9). Output can be directed to a printer or a file; it is not automatically displayed at the terminal. Submitting one batch job from another batch job is also called deferred batch (refer to the NOS Reference Manual, Volume 1).

CONVERSATIONAL BATCH

Conversational batch refers to the BATCH subsystem available under time-sharing job processing (refer to section 9). Under this subsystem, the terminal user can enter individual batch control statements and receive immediate execution of this statement. This use of batch control statements is classed as part of a time-sharing job and not a batch job.

BATCH JOB STRUCTURE

The user organizes his batch job according to the following requirements.

1. The job must be identified.
2. The user must be identified.†
3. A charge number must be specified.†
4. The entire job must be divided into records.
5. The first record must contain all the control statements.
6. The end of the job must be identified.

A model batch job punched on cards is shown in figure 3-1. All the control statements, and only the control statements, appear in the first record. A job may consist solely of this single record of control statements.

The first statement of the control statement record must be the job statement. If the installation has user validation, the second statement must be the USER statement. If the installation employs user accounting control, the third statement must be the CHARGE statement. These statements are explained later in this section. The remainder of the statements in the control statement record direct processing of the job.

The records that follow the control statement record are programs and data. The data can be read by the programs or by control statements. The program and data records must appear as they are referenced (refer to figure 3-1). The particular control statements shown are explained later in this guide.

†This is an optional installation requirement. It is the usual case.

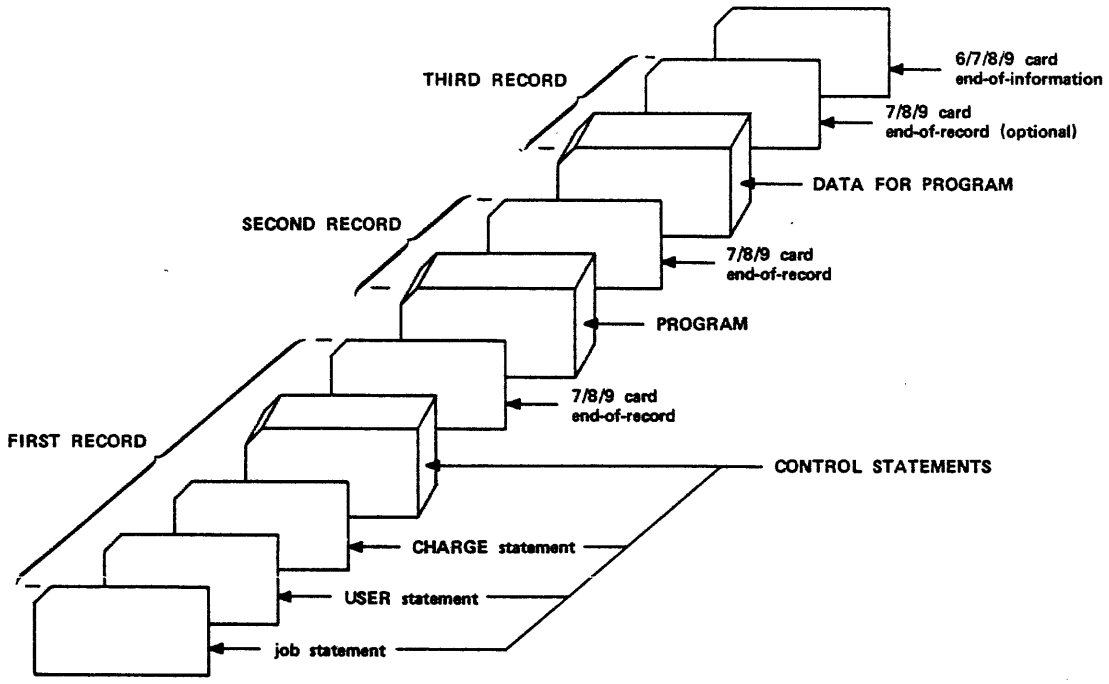


Figure 3-1. Batch Job Representation

CONTROL STATEMENT FORMAT

Generally, a control statement can contain up to 80 characters. Ordinarily, it may begin in any column but must end by column 80.

Most control statements begin with a name that suggests the nature of the operation to be performed (COPY, REWIND, SAVE). Exceptions are control statements beginning with an asterisk or a number. The asterisk signals a line of comment. Numbered control statements are explained in section 6.

A control statement usually requires a parameter list to specify the file names, options, and numerical values which will be used in the operation the statement initiates. This list follows the control statement name on the same line. Separators must be inserted between the parameters. The principal separators are the following.

, (

These separators can be used interchangeably unless a format requires a specific separator.

Spaces in a control statement are ordinarily ignored. Exceptions are the job statement and the USER statement which must not contain any spacing.

The following examples are acceptable control statements.

```
USER(XXX,YYY)
G E T    (CON T   ROL   )
   CO    PY,   CONTROL   ,   BETA   )
    REWIND   ,   BETA   .
```

The parameters included with a control statement are either order-independent (appear in any order) or order-dependent (must appear in a specified order).

Parameters are order-independent if they are specified with a keyword. A keyword is an alphanumeric mnemonic with a defined meaning for the system. Some keywords have meanings by themselves; other keywords require the user to supply values (usually with an =).

Parameters are order-dependent if they are user-supplied names or values whose application is identified by their location in the list following the control statement name. If an order-dependent parameter is omitted, the separator that would have followed it must be included. This is necessary so the system can match the remaining parameters in the list with their appropriate application. If no parameters remain, the additional separator is not necessary.

A few control statements allow order-independent and order-dependent parameters in the same list. In this format, the order-dependent parameters must come first. Some control statements have both an order-independent and an order-dependent version. In this guide, order dependency is specified in the control statement descriptions.

In many instances, the system will supply a default value for a missing parameter. This will be the value commonly used for that parameter. Defaults of interest to the user are identified in the control statement descriptions.

Every control statement must end with a terminator on the same line. A terminator can be either of the following.

) .

Examples:

The operation of the following control statements is explained in later sections; however, the basic action is outlined to illustrate control statement format.

The following is an order-dependent control statement.

COPYBR(INPUT,MYFILE,1)

This copies one record from the INPUT file to a user-created file, MYFILE. Since INPUT is the default for the first parameter and 1 is the default for the third parameter, this statement could be written as follows:

COPYBR(,MYFILE)

The following is an order-independent control statement.

PURGE(ALPHA,FRED,TEST)

This removes three user-created permanent files from the system.

The following is an order-independent control statement.

COBOL5(I=TAPE1,B,L=TEMP)

This initiates the compilation of source code read from a file TAPE1. The binary output from compilation is written on the system file BIN (B), and the source listing and diagnostics are written on the user-specified file TEMP.

CONTROL STATEMENT RECORD

The first record of every job must be composed exclusively of all the control statements for the job. This is called the control statement record. It is this record which controls the details of job processing.

The remainder of this section discusses the five principal control statements of a control statement record.

JOB CONTROL STATEMENT

A job statement or job card must begin every control statement record. This statement identifies the job with a user-chosen name and, optionally, specifies a job step time limit and/or maximum storage requirement. The beginning batch user need only be concerned with specifying the name. The remaining parameters and job step definitions are explained in the NOS Reference Manual, Volume 1.

The basic format of the job statement is as follows:

jobname.

jobname	A 1- to 7-character user-chosen alphanumeric job name which must begin with a letter.

The following examples are valid job statements.

MYJOB.
A.
JONES.
JOB2345.

USER CONTROL STATEMENT

A USER statement must follow the job statement if the installation requires validation. The USER statement establishes:

- That the programmer submitting this job is a legal system user
- What system resources the programmer may use and to what extent he may use them (appendix C)
- The location of his permanent files (section 5)

For all batch jobs, the USER statement must immediately follow the job statement.

Format of the order-dependent USER statement is as follows:

USER(usernum,passwor,family)

usernum	User number given the programmer by the installation if validation is required.
passwor	Optional password which the user gives himself or is assigned (appendix C).

family	Identifies a family name. It is included only if the installation is using family names to group permanent file devices. If the installation is using family names and this parameter is not specified, the user is assigned to the default family.

Example:

```
USER(EFD2501,APRIL)
```

The installation has given this user the user number EFD2501 and has assigned him the password APRIL. Family names are either not in use or if they are, this user will be assigned to the default family.

CHARGE CONTROL STATEMENT

If an installation is charging individual users for system resources used, it will give them charge number/project number combinations which must be included with their jobs. In such a case, a user must add a CHARGE statement immediately after his USER statement to access the system. Format of the order-dependent CHARGE statement is as follows:

```
CHARGE(chargenum,projectnum)
```

chargenum	A 1- to 10-character alphanumeric charge number the installation gives the user
projectnum	A 1- to 20-character alphanumeric project number the installation may or may not give the user

Example:

```
CHARGE(510,299N9)
```

The installation has given this user the charge number 510, and the user is charging this job to the project designated 299N9.

CONTROL STATEMENT RECORD COMMENTS

A programmer can insert descriptive comments in a control statement record in the following manner. These comments will appear in the dayfile (appendix B) at the end of the printout from job processing.

- Place the comment after the terminator of most control statements
- Place the comment after the terminator of a COMMENT statement
- Place the comment after a leading *

Formats of a comment on a control statement are as follows:

```
name(parameters)comment

name.comment
```

Format of a comment on a COMMENT statement is as follows:

```
COMMENT.comment
```

name	Name of the control statement
parameters	Parameter list that goes with the control statement
comment	Character string that can begin in any column after the terminator but cannot go beyond column 80

Format of a comment after an * is as follows:

```
*comment
```

The * can be in any column, but it must be the first nonblank character. The comment is a character string that can begin in any column after the * but cannot go beyond column 80.

The dayfile format is 40 columns wide (appendix B). Comments which go beyond column 40 in the control statement record will appear as two lines in the dayfile. If a comment begins after column 40 on a control statement, it appears in the dayfile as two lines (a blank line followed by a line containing all the characters following column 40).

A job demonstrating comments is shown in figure 3-2. The dayfile printed when this job is processed is shown in figure 3-3.

```
EFD.
USER(EFD25,)
CHARGE(59,N4)
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
COMMENT.THIS COMMENT IS TOO LONG TO BE COMPLETED BY COLUMN 80 AND SO AN ASTERISK
*MUST BE USED TO BEGIN THE NEXT LINE.
LIMITS. THIS COMMENT FOLLOWS THE TERMINATOR OF A CONTROL STATEMENT.
*                                       THIS COMMENT BEGINS IN COLUMN 41
COMMENT.    THIS COMMENT IS CONTINUED TO AN ADDITIONAL 2 LINES.  THE FINAL LINE
*  DOES NOT FOLLOW AN ASTERISK, A COMMENT STATEMENT, OR A CONTROL STATEMENT AND
THEREFORE IT SHOULD PRODUCE AN ERROR MESSAGE AND JOB TERMINATION.
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
6/7/8/9
```

Figure 3-2. Comment Demonstration

```
15.38.42.EFD
15.38.42.USER(EFD25,)
15.38.42.CHARGE(59,N4)
15.38.42.*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15.38.42.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15.38.42.COMMENT.THIS COMMENT IS TOO LONG TO BE C
15.38.42.OMPLETED BY COLUMN 80 AND SO AN ASTERISK
15.38.42.*MUST BE USED TO BEGIN THE NEXT LINE.
15.38.43.LIMITS. THIS COMMENT FOLLOWS THE TERMINA
15.38.43.TOR OF A CONTROL STATEMENT.
15.38.43.*
15.38.43.THIS COMMENT BEGINS IN COLUMN 41
15.38.43.COMMENT.    THIS COMMENT IS CONTINUED TO
15.38.43. AN ADDITIONAL 2 LINES.  THE FINAL LINE
15.38.43.*  DOES NOT FOLLOW AN ASTERISK, A COMMEN
15.38.43.T STATEMENT, OR A CONTROL STATEMENT AND
15.38.43.THEREFORE IT SHOULD PRODUCE AN ERROR MES
15.38.43.SAGE AND JOB TERMINATION.
15.38.43. FORMAT ERROR ON CONTROL CARD.
```

Figure 3-3. Printed Dayfile After Processing

PROGRAMS

A user can include programs in his job as separate records after the control statement record. If these programs require data input, that too can be added as separate records. The user initiates compilation of these programs by inserting language processor call statements in the control statement record. Placement of these language statements matches the appearance of the program records in the job. The field length necessary for compilation is set by the system.

This guide outlines the language statements for BASIC, FORTRAN, COBOL 4, and COBOL 5. The formats of these language statements are as follows:

BASIC(parameters)
FTN(parameters)
COBOL(parameters)
COBOL5(parameters)

The common parameters for these formats are outlined in table 3-1. Parameters with default values (designated as omitted) are adequate for ordinary use. The full set of defaults for a particular language statement is in effect when the name and a terminator are used as follows:

BASIC.
FTN.
COBOL.
COBOL5.

For BASIC, the default is automatic execution after compilation. For FTN and COBOL, the default is no program execution after compilation. If the user intends to execute these programs, he must add a statement containing the name of the file to which the object code was written. If the file LGO is used by default, the format is as follows:

LGO.

This rewinds the compiled (binary) file, loads it into memory, and executes it. A compiled FORTRAN program will also execute if a GO has been included in the parameter list of the language statement (refer to table 3-1).

As an alternative, the user can put the compiled program on a file he names with the following parameter.

B=lfn

To execute the compiled program lfn, he follows it with the following statement.

```
lfn.
```

A REWIND statement is unnecessary, since binary files are automatically rewound.

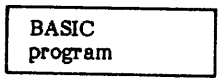
Refer to the NOS Reference Manual, Volume 1 for a complete list of parameters and defaults for each of these language statements.

Example 1:

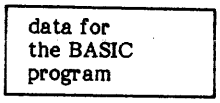
The following job compiles and executes a FORTRAN and a BASIC program.

```
JOB1.
USER statement
CHARGE statement
FTN.
LGO.
BASIC.
end-of-record
```


```
end-of-record
```




```
end-of-record
```



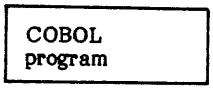

```
end-of-information
```

The FTN statement initiates compilation of the next record in the job, the FORTRAN program. The LGO rewinds, loads, and executes the compiled program. The BASIC statement initiates compilation and execution of the next record in the job, the BASIC source program. Execution of the BASIC program calls for data input, which comes from the next (and last) record in the job.

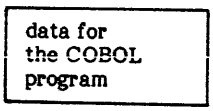
Example 2:

The following job compiles a COBOL program, puts the object code on a user file, and then executes that file.

```
JOB2.
USER statement
CHARGE statement
COBOL(B=TEMP)
TEMP.
end-of-record
```




```
end-of-record
```




```
end-of-information
```

The COBOL statement initiates compilation of the next record in the job, a COBOL program. The object code is put on a user-designated file which he calls TEMP. To execute this object program, the user enters the file name as a control statement. The COBOL program requires data input, which comes from the next and last record in the job.

TABLE 3-1. COMMON PARAMETERS FOR LANGUAGE PROCESSOR CALL STATEMENTS

Parameter	BASIC	COBOL	COBOL5	FTN
I	Input on COMPILE†	Input on INPUT	Input on COMPILE†	Input on COMPILE†
I=lfn	Input on lfn			
I omitted	Input on INPUT			
B	Object code on BIN	Object code on LGO	Object code on BIN	Object code on LGO
B=lfn	Object code on lfn			
B omitted	Compile-to-memory (no object code)	Object code on LGO		
L	Output on OUTPUT		Output on LIST	Output on OUTPUT
L=lfn	Output on lfn			
L omitted	Batch: output on OUTPUT Time sharing: no output	Output on OUTPUT		
GO	Object code loaded and executed	N/A		Object code loaded and executed
GO omitted	Compile-to-memory	N/A		Object code not loaded and executed

†Not covered in this guide.

This section explains the creation and characteristics of local files. It also describes the copy and file positioning control statements which are used with files local to a job. The following control statements are included in this section.

COPYBR	VERIFY
COPYBF	SKIPR
COPYEI	SKIPF
COPY	SKIPEI
COPYCR	BKSP
COPYCF	SKIPFB
COPYSBF	REWIND

Unless he explicitly requests otherwise, the files a user's job creates are intended for the temporary use of the job. When job processing terminates, these temporary files are released. Such files are local files. Local files are distinguished from permanent files (section 5) which remain a part of the system when job processing terminates.

A job can create local files in the following ways.

- Using the name of the file for the first time in a copy statement (refer to Copy Statements)
- Through program execution
- Making a copy of an existing permanent file (refer to section 5)

In the parameter lists of the control statement descriptions, a local file is identified by lfn (local file name).

The programmer, in his manipulations of files, may use the copy and file positioning control statements (refer to Copy Statements and File Positioning Statements).

COPY STATEMENTS

Copy control statements make copies of files and records. The following parameters apply to all copy statements.

lfn_1 Source file of the copy operation; default is INPUT.

lfn_2 File which will be the copy; default is OUTPUT.

n Number of records or files to copy; default is 1.

All copy operations begin copying from file lfn1 at its current position and begin copying to the current position of lfn2 (an exception is the COPYEI statement). If lfn2 does not exist, it is created by the system. If lfn1=lfn2, no copy takes place, and n records or files are skipped. After the copy operation, subsequent accesses of either file may begin where the copying stopped, at the BOI, or where the user has positioned the file. This depends upon the parameters included with the copy statement or intervening file positioning statements (refer to Binary Coded Statements).

BINARY COPY STATEMENTS

Binary copy statements copy on a bit-by-bit basis and produce a duplicate of binary and, in most cases, coded data.

The following statements are the principal binary copy statements.

COPYBR(lfn_1,lfn_2,n)

This copies n records from lfn_1 to lfn_2. If the EOI on lfn_1 is encountered before n records are copied, an EOF is added to lfn_2, and the operation terminates.

COPYBF(lfn_1,lfn_2,n)

This copies n files from lfn_1 to lfn_2. If more than one file is copied, the result is a multifile file. If the EOI on lfn_1 is encountered before n files are copied, an EOF is added to lfn_2, and the operation terminates.

COPYEI(lfn_1,lfn_2,x)

This copies file lfn_1 to file lfn_2 until an EOI is encountered on lfn_1. If a third parameter (x) is present, both files are rewound before the copy and then, after the copy is completed, both files are rewound, verified, and rewound. This x parameter can be any 1- to 7-character alphanumeric name. The verification is a bit-by-bit comparison of the copy with the original. Errors, if found, are identified in the job output.

The following copy statements in a job create three new files from two old files (figure 4-1).

```
COPYBF(ORIG1,NEW1)
COPYBR(ORIG2,NEW2)
COPYBR(ORIG2,NEW3)
```

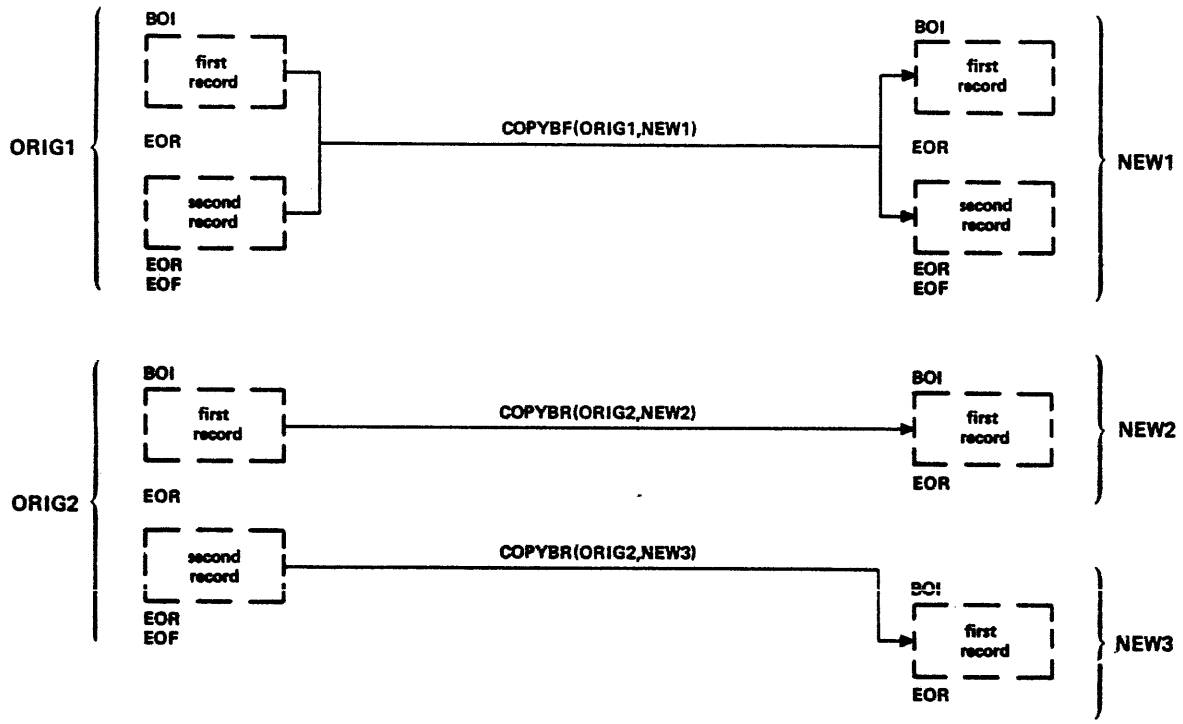


Figure 4-1. File Creation with Copy Statements

CODED COPY STATEMENTS

Coded copy statements copy on a character-by-character basis. They treat the characters as constituting line images of up to 150 characters per line. Accordingly, the coded copy statements only duplicate information that is in coded lines (program output, text files, and so on).

Primarily, a coded copy is employed to reformat a coded file. The following statements are coded copy statements.

COPYCR(lfn_1,lfn_2,n,fchar,lchar)

This copies n coded records from lfn_1 to lfn_2. If fchar and/or lchar are present, they specify the first and last character of each line where copying begins and ends. This vertically truncated version of the original is left-justified in the copy. If fchar and lchar are both omitted, no reformatting takes place. If the EOI or EOF is encountered before n records are copied, an EOF is added to lfn_2 and the copy operation terminates.

COPYCF(lfn_1,lfn_2,n,fchar,lchar)

This copies n coded files from lfn_1 to lfn_2. If n is greater than 1, these files are multifiles. If fchar and/or lchar are present, they specify the first and last characters of each line where copying begins and ends. This vertically truncated version of the original is left-justified in the copy. If fchar and lchar are both omitted, no reformatting takes place. If the EOI is encountered before n files are copied, an EOF is added to lfn_2, and the copy operation terminates.

COPYSBF(lfn_1,lfn_2,n)

This copies n files from lfn_1 to lfn_2, shifting each line one character to the right and adding a leading space. (A space is the printer carriage control character for single spacing.)

The character 1 is inserted at the beginning of each record as the files are copied. (A 1 is the printer carriage control character for starting a new page.)

If an EOI is encountered before n files are copied, an EOF is added to lfn_2, and the operation terminates.

This copy statement is primarily used to format the OUTPUT file for printing. It is possible to copy INPUT directly to OUTPUT. Using defaults, the control statement would be as follows:

```
COPYSBF.
```

The following list is a segment from an employee file called EMPLOY3.

1445	JONES, E. T.	J22	66.25	GS7
1446	KING, L. A.	K921	49.88	GS7
1447	LANG, R. B.	L404	43.35	GS7
1448	LONG, T. M.	L11	38.97	GS11

This segment gives the sequence number, name, employee number, deductions, and rating for individual employees. To create a second file EMPLOY4 with only the name and employee number, use the following statement.

```
COPYCF(EMPLOY3,EMPLOY4,1,7,30)
```

This statement extracts the characters from columns 7 through 30 and left-justifies them in the new file EMPLOY4. This file is as follows:

```
JONES, E. T.     J22
KING, L. A.      K921
LANG, R. B.      L404
LONG, T. M.      L11
```

VERIFY STATEMENT

The VERIFY statement makes a bit-by-bit comparison of two files to detect any variance between the contents of these files. This is a convenient way to check for errors in a copy operation. The basic format of this statement is as follows:

VERIFY(lfn_1,lfn_2,p_1,p_2,...p_n)

lfn_1 and lfn_2 — Two files being compared.

p_i — Defines the comparison with the following values.

p_i	Description
N=0	Verify terminates on the first empty file encountered on either file.
N=x	Verify x files; default is 1.
N	Verify terminates when EOI is encountered on either file.
R	Rewind both files before and after the verify.
A	Abort if error occurs.

(Refer to the NOS Reference Manual, Volume 1 for the full range of p_i values.)

As an example, the following job demonstrates the operation of the VERIFY statement. Two input files with slight differences are copied into the system. The VERIFY statement is used to detect these differences.

```
DEMO.
USER Statement
CHARGE Statement
COPYBR(,FILE1)
COPYBR(,FILE2)
VERIFY(FILE1,FILE2,R)
7/8/9
$608.20
$012.33
$111.49
7/8/9
$608.20
$012.34
$101.49
6/7/8/9
```

The output from the verification is shown in figure 4-2. The first word was a match; the next two words were not and were listed in octal display code with the logical difference of each pair to pinpoint the location in the word where the difference was found. The first error is diagrammed to show how this is read.

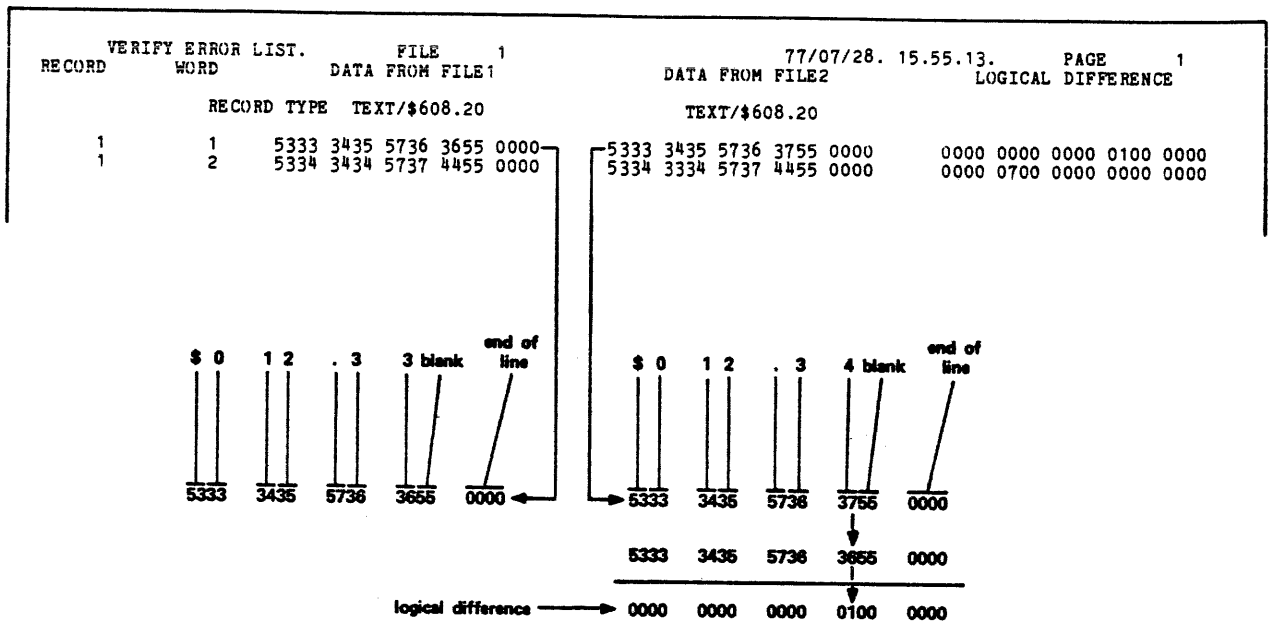


Figure 4-2. Output from Processing a VERIFY Statement

FILE POSITIONING STATEMENTS

File positioning control statements reposition the current position of a file for selective reading, writing, and copying. Only the principal parameters are explained; the full set of parameters for each statement is given in the NOS Reference Manual, Volume 1.

BACKGROUND

The current position of a file is the location in the file where the next operation that accesses the file will begin to read or write data. This location can be at the BOI, after an EOR, after an EOF, or at the EOI. Reading records from a file can leave the current position at the end of the last record read, writing a file with a program can leave the current position at the end of the last record written, and copying statements can leave the current position of both the original and the copy after the last record or file copied.

FORWARD POSITIONING STATEMENTS

The following statements move the current position of a file forward toward the EOI.

SKIPR(lfn,n)

This moves the current position of the named file (lfn) forward the designated number of records (n). EOF marks are separate records and are included in the record count. If the EOI is encountered before n files are passed, the current position remains at the EOI.

SKIPF(lfn,n)

This moves the current position of the named multifile file (lfn) forward the designated number of files (n). If the EOI is encountered before n files are passed, the current position remains at the EOI.

SKIPEI(lfn)

This moves the current position of the named file or multifile file (lfn) to EOI. On magnetic tapes, which have no EOI, the operation stops at the EOF indicator.

BACKWARD POSITIONING STATEMENTS

The following statements move the current position of a file backwards toward the BOI.

BKSP(lfn,n)

This moves the current position of the named file (lfn) backwards the designated number of records (n). EOF indicators are separate records and are included in the record count. If BOI is encountered before n records are passed, the current position remains at the BOI.

SKIPFB(lfn,n)

This moves the current position of the named multifile file (lfn) backwards the designated number of files (n). If BOI is encountered before n files are passed, the current position remains at the BOI.

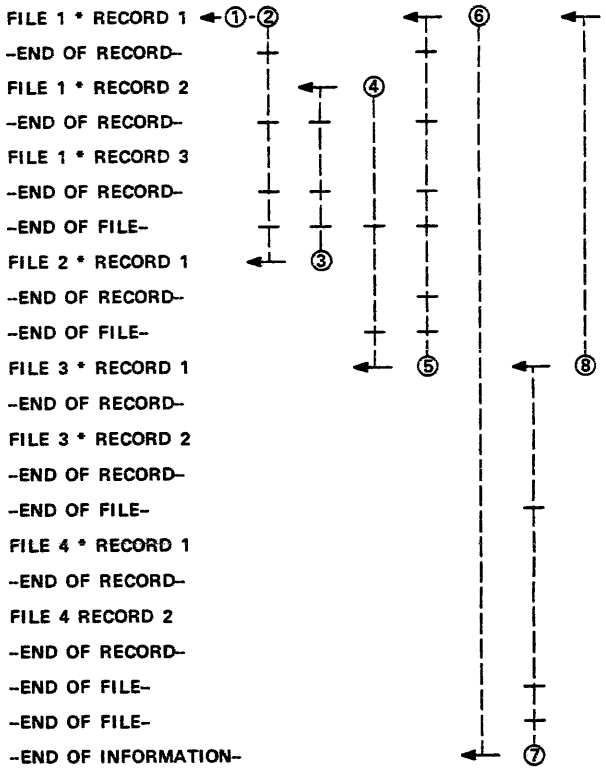
REWIND(lfn_1,lfn_2,...)

This moves the current positions for files lfn_1,lfn_2,... back to the BOI.

The following job is made up of file positioning statements and one-line records organized into files. This job demonstrates the operation of the positioning statements.

```
job statement
USER statement
CHARGE statement
COPYEI(,DEMO)
REWIND(DEMO)
SKIPR(DEMO,4)
BKSP(DEMO,3)
SKIPF(DEMO,2)
BKSP(DEMO,10)
SKIPEI(DEMO)
SKIPFB(DEMO,3)
REWIND(DEMO)
7/8/9
FILE 1 * RECORD 1
7/8/9
FILE 1 * RECORD 2
7/8/9
FILE 1 * RECORD 3
7/8/9
6/7/9
FILE 2 * RECORD 1
7/8/9
6/7/9
FILE 3 *RECORD 1
7/8/9
FILE 3 * RECORD 2
7/8/9
6/7/9
FILE 4 * RECORD 1
7/8/9
FILE 4 * RECORD 2
7/8/9
6/7/9
6/7/8/9
```

Figure 4-3 shows the structure of this multifile file in memory and how the position is moved by the operation of each control statement. The crossmarks on the broken lines show the terminators that were counted in the operation. If this multifile file had had only one EOF before the EOI, operation 7 would have moved the position to FILE 2 * RECORD 1.



NOTES:

①	REWIND(DEMO)	⑤	BKSP(DEMO,10)
②	SKIPR(DEMO,4)	⑥	SKIPEI(DEMO)
③	BKSP(DEMO,3)	⑦	SKIPFB(DEMO,3)
④	SKIPF(DEMO,2)	⑧	REWIND(DEMO)

Figure 4-3. Operation of File Positioning Statements on a Multifile File

This section explains how a user can create files within the system and preserve them beyond job processing. The following control statements are described in this section.

SAVE	ATTACH
GET	CHANGE
REPLACE	PURGE
APPEND	PERMIT
DEFINE	CATLIST

Only the principal parameters are explained; the complete set of parameters for each control statement is given in the NOS Reference Manual, Volume 1.

A user can create, access, and modify mass storage files that remain within the system until he or the installation specifically removes them. These files are called permanent files.

Permanent files are either indirect or direct access according to the manner in which they are accessed. An indirect access permanent file is accessed by making a local copy and having control statements operate on that copy rather than the original. This copy will be released after job processing unless the user specifically saves it as an added permanent file or uses it to replace the original version. A direct access file is accessed in place; that is, control statements interact with the permanent file. No copy is made.

Direct access permanent files offer a variety of multiple access (simultaneous access of a single file by two or more users) capabilities not available with indirect access files (refer to section 8 of the NOS Reference Manual, Volume 1). Also, direct access files have greater I/O efficiency, because no intermediate copy is used. For the time-sharing user, an advantage of the direct access file is that all modifications and additions are made directly on the permanent file and should the terminal become disconnected or the system go down, all previous work is not lost. However, care must be taken in altering a direct access file since all alterations (correct and erroneous) are made directly to the permanent file, not a copy.

Indirect access files require smaller allocations of mass storage space than direct access files. Thus, permanent files that are smaller in size are typically made indirect access files. The character content of this section on permanent files is a good measure of the upper limit for an indirect access permanent file.

INDIRECT ACCESS PERMANENT FILES

HOW TO CREATE AN INDIRECT ACCESS PERMANENT FILE

The data that is to be made an indirect access permanent file must first exist as a local file. This local file may be created in any manner by job processing. The system copies the local file to the permanent file space on mass storage when the user includes the following control statement in a control statement record.

SAVE(lfn)

lfn Name of the local file

The permanent file copy will have the same name. If the user wants the permanent file to have a different name, he uses the following form.

SAVE(lfn=pfn)

pfn Name the user gives the indirect access permanent file. This name must be used for future access of this permanent file.

The SAVE statement copies the entire file; that is, the permanent file will contain the entire local file whatever the current position of lfn when the SAVE statement is processed. After SAVE is processed, the local file is rewound.

In the following example, an input record is made a local file with the name A. This local file is made an indirect access permanent file with the same name.

```
COPYBR(,A)
SAVE(A)
```

A job contains the following control statements.

```
BASIC(B=ZETA)
SAVE(ZETA)
```

The BASIC statement compiles a BASIC source program, which is a record in the same job. The binary object code is put on a local file with the name ZETA. The SAVE statement makes this local file an indirect access permanent file with the same name.

HOW TO ACCESS AN INDIRECT ACCESS PERMANENT FILE

A local copy of an indirect access permanent file can be obtained by the creator of the permanent file if he includes the following control statement in the control statement record of a job he submits.

GET(pfn)

If the user wants the local copy to have a different name, he enters:

GET(lfn=pfn)

lfn — Temporary name the user has chosen for the local copy to be made from pfn

The local copy is always at the BOI after GET is processed. If a local file named lfn existed before the GET request, it is lost when GET is processed.

Example 1:

To obtain a local copy of the indirect access permanent file A created in a previous example, the user enters the following control statement.

GET(A)

If he wants the local copy to have the name INFILE, he enters:

GET(INFILE=A)

Example 2:

To execute the BASIC program compiled and saved in a previous example, the user includes the following control statements in a job.

```
GET(ZETA)
ZETA.
```

HOW TO ADD INFORMATION TO AN INDIRECT ACCESS PERMANENT FILE

The user can make additions to one of his indirect access permanent files if the additions are first made local files. He adds these local files to the permanent file with the following APPEND control statement.

APPEND(pfn,lfn_1,lfn_2,...)

pfn — Name of the indirect access permanent file to receive the additions

(lfn_1,lfn_2,...) — Local files that constitute the additions

For example, a user keeps a weekly journal of transactions. This is an indirect access permanent file called JOURNAL. Approximately every day he adds new transactions to this file. To add the transactions for Thursday and Friday, which have been made into two local files called THUR and FRI, he enters the following control statement.

APPEND(JOURNAL,THUR,FRI)

HOW TO MODIFY AN INDIRECT ACCESS PERMANENT FILE

If the user wants to modify one of his indirect access permanent files, he can do one of the following.

- Get a local copy, alter the local copy with positioning and copy statements or with a program, and then replace the existing permanent file with this new version
- Produce a new local file and, without getting a copy of the existing permanent file, replace the permanent file with the new local copy

In either case, the REPLACE control statement replaces the existing pfn with the new lfn. If pfn and lfn have the same name, the format of the statement is as follows:

REPLACE(pfn)

If the local copy has a different name, the format is as follows:

REPLACE(lfn=pfn)

For example, a firm maintains a file of 18 part listings. Each listing is a single record in the file. The file is an indirect access permanent file called PARTLST. At a particular date, this file will be updated with one deletion and two additions. The updated file will have 19 part listing records; the job is as follows:

```
job statement
USER statement
CHARGE statement
GET(LOCAL1=PARTLST)
COPYBR(LOCAL1,LOCAL2,5)
COPYBR(,LOCAL2,1)
COPYBR(LOCAL1,LOCAL2,7)
SKIPR(LOCAL1,1)
COPYBR(LOCAL1,LOCAL2,3)
COPYBR(,LOCAL2,1)
COPYBR(LOCAL1,LOCAL2,2)
REPLACE(LOCAL2=PARTLST)
—end-of-record—
```

```
First Addition
```

```
—end-of-record—
```

```
Second Addition
```

```
—end-of-information
```

Figure 5-1 outlines the job processing that creates the updated file. A local copy (LOCAL1) of the indirect access permanent file PARTLST is made. By copying from this file and the INPUT records, a second local file (LOCAL2) is created. This file has one deletion and two additions. Finally, LOCAL2 replaces the existing permanent file PARTLST. Future access of PARTLST will receive a copy of this updated version.

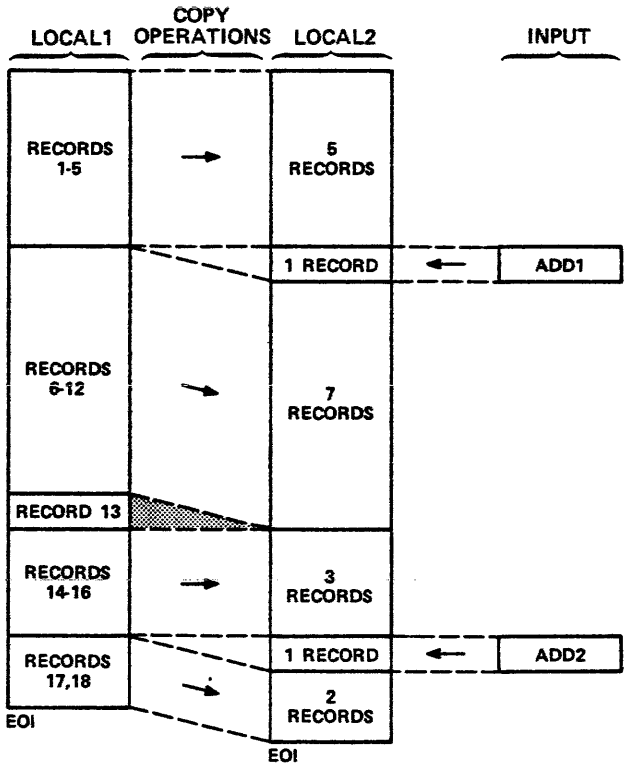


Figure 5-1. File Update

DIRECT ACCESS PERMANENT FILES

HOW TO CREATE A DIRECT ACCESS PERMANENT FILE

A user can create a direct access permanent file by defining an area of mass storage for that purpose with the DEFINE control statement and then copying a local file or input information to that area. The basic format of the DEFINE control statement is as follows:

```
DEFINE(pfn)
```

pfn Name the direct access file will have

In the following example, the first statement defines an area on mass storage for a direct access file with the name FILE24, and the second statement copies an input file to this area.

```
DEFINE(FILE24)
COPYBF(,FILE24)
```

The following example defines a direct access file with the name NEWFILE and obtains local copies of two indirect access permanent files (TEST1 and TEST2). TEST1 is positioned to the fourth record, and the fourth, fifth, and sixth records are copied to NEWFILE. TEST2 is positioned to the third record, and the third and fourth records are added to NEWFILE with a copy. NEWFILE is rewound and copied to the printer for verification.

```
DEFINE(NEWFILE)
GET(TEST1,TEST2)
SKIPR(TEST1,3)
COPYBR(TEST1,NEWFILE,3)
SKIPR(TEST2,2)
COPYBR(TEST2,NEWFILE,2)
REWIND(NEWFILE)
COPYSBF(NEWFILE,)
```

HOW TO ACCESS A DIRECT ACCESS PERMANENT FILE

A user can access a direct access permanent file he has created by including an ATTACH control statement in a job he submits for processing. The basic format of this statement is as follows:

ATTACH(pfn)

pfn Name of the direct access permanent file being accessed

After processing this statement, control statements that follow in the same job can interact directly with the contents of this file.

If the user wants to access the direct access file with a different name during job processing, he employs the following format.

ATTACH(lfn=pfn)

lfn Temporary substitute name

HOW TO MODIFY A DIRECT ACCESS PERMANENT FILE

The copy and file positioning control statements can be employed by a programmer to alter his direct access files. However, if the programmer is periodically updating large files, he should investigate the capabilities offered by the system utility control statements (refer to section 14, NOS Reference Manual, Volume 1).

Whenever the user initiates alterations of a direct access permanent file, he makes changes to the permanent file, not a temporary copy. As a precaution against inadvertent modification of a direct access file, the system requires the user to specify write mode on an ATTACH statement before he can alter the file attached. The format of an ATTACH statement that grants write privileges is as follows:

ATTACH(lfn=pfn/M=W)

M=W Signifies mode equals write

Example 1:

The following job adds a new record to an existing direct access permanent file called LISTING.

```
job statement
USER statement
CHARGE statement
ATTACH(LISTING/M=W)
SKIPEI(LISTING)
COPY(,LISTING)
7/8/9
 new
 record
6/7/8/9
```

Example 2:

The following control statements replace the fifth record of a direct access file.

```
ATTACH(NEWFILE/M=W)
SKIPR(NEWFILE,4)
COPYEI(NEWFILE,TEMP)
REWIND(NEWFILE,TEMP)
SKIPR(NEWFILE,4)
COPYBR(,NEWFILE)
SKIPR(TEMP)
COPYEI(TEMP,NEWFILE)
```

These statements attach the direct access file NEWFILE in write mode, skip four records, and copy the remainder of the file to a temporary file called TEMP. A new input record is copied to NEWFILE as a fifth record. The first record in TEMP is skipped, and the remainder of TEMP is returned to NEWFILE.

PURGING PERMANENT FILES

The user can purge (remove from the system) one or more of his indirect and direct access permanent files by including the PURGE control statement in a job. The basic format of this statement is as follows:

PURGE($file_1$,$file_2$,...)

$file_1$,$file_2$,... Names of the user's permanent files to be removed from the system. The removal is permanent, and they can in no way be accessed in the future.

ALTERNATE ACCESS OF PERMANENT FILES

A user can grant other users access permission to his permanent file by selecting an appropriate access category when he creates the file. He selects this category with an added parameter on the SAVE or DEFINE statement. This parameter is a keyword with the following form.

CT=ct

ct One of the following access categories of a permanent file.

P Private file. This limits access to the originator of the file unless he specifies individual alternate users with the PERMIT control statement (explained later). Private is the default category for permanent files.

S — Semiprivate file. This limits access to the originator and other users who include the following parameters on a GET or ATTACH control statement.

- Name of the file
- User number of the originator
- Password specified by the originator for this file (originator's option)

The system keeps a detailed record of all alternate accesses to this file. This record is available with the CATLIST control statement (explained later).

PU — Public file. Access requirements are identical to the semiprivate file; however, the system keeps a record only of the total number of accesses.

In addition to establishing an access category, to determine who can access the file, the originator can establish an access mode to determine how an alternate user can access the file. Mode is specified on the SAVE or DEFINE control statement with the following keyword parameter.

M=m.

This guide considers only two values of m, R (read) and W (write). Eight modes are available and are explained in the NOS Reference Manual, Volume 1.

If read is specified with M=R, alternate users can only read the file after accessing it. If write is specified with M=W, alternate users can read and write after accessing the file.

The originator of an indirect access permanent file automatically has read and write permission when he accesses that file. For a direct access file, the default mode is read, and the user must specify M=W if he intends to write on it.

The originator of a permanent file can give that file greater security by associating a password with it. This password has no relation to the password included on the USER control statement, which gives access to the system. The file password is specified on a SAVE or DEFINE statement with the following keyword parameter.

PW=passwor

passwor — A 1- to 7-alphanumeric character password

If a user wants to create an indirect access permanent file that can be accessed by others, he uses the following format of the SAVE statement.

SAVE(lfn=pfn/PW=passwor,CT=ct,M=m)

An alternate user who wants a copy of this file includes the following format of the GET control statement in his job.

GET(lfn=pfn/PW=passwor,UN=usernum)

usernum — User number of the file's originator

The keywords PW=, CT=, M=, and UN= appear in any order after the slash. The slash is a required separator that must come after the file names if any keywords are to be used.

Example 1:

A user with user number AB22 creates an indirect access permanent file called DATA8. He gives it the password ABC, makes it public, and grants read mode to alternate users. The format of the SAVE statement with which he does this is as follows:

SAVE(DATA8/PW=ABC,CT=PU,M=R)

If the originator of this file wants a copy, he uses the following statement.

GET(DATA8)

If an alternate user wants a copy of this file, he inserts the following statement in his job.

GET(DATA8/UN=AB22,PW=ABC)

If the originator of a permanent file has made it private by default or specification, he can grant alternate access only to users he specifies on a subsequent PERMIT statement. Format of this statement is as follows:

PERMIT(pfn,$usernum_1$=m_1,$usernum_2$=m_2...)

$usernum_i$ — User numbers of those who are being granted access permission to the private file with the name pfn

m_i — Permission modes R (read) and W (write)

Example 2:

A user with user number AB22 includes the following control statements in a job.

SAVE(LIST28)
PERMIT(LIST28,SM182=R)

SAVE creates an indirect access permanent file LIST28 with private category and no password. The permit grants read access to the alternate user with user number SM182.

To obtain a copy of LIST28, the alternate user includes the following control statement in his job.

```
GET(LIST28/UN=AB22)
```

If the user wants to create a direct access permanent file that can be accessed by others, he uses the following format of the DEFINE control statement.

```
DEFINE(lfn=pfn/PW=passwor,CT=ct,M=m)
```

An alternate user who wants to access this file includes the following format of the ATTACH statement in his job.

```
ATTACH(lfn=pfn/PW=passwor,UN=usernum,M=m)
```

lfn — Optional substitute name for the permanent file name pfn

UN — User number of the originator of the file

The default mode is M=R. If the alternate user has been granted write permission for this file and wants to exercise that permission, he must include M=W, as the originator must do.

Example 3:

A user with user number AA29A creates a direct access file called XX7X. He assigns the password ENTER, establishes the category as semiprivate, and specifies write permission for alternate users with the following statement.

```
DEFINE(XX7X/PW=ENTER,CT=S,M=W)
```

If the originator of this file wants to access it, he uses the following control statement.

```
ATTACH(XX7X)
```

If the originator wants to write on it, he uses the following statement.

```
ATTACH(XX7X/M=W)
```

If an alternate user wants to access this file for reading, he includes the following control statement in his job.

```
ATTACH(XX7X/PW=ENTER,UN=AA29A)
```

If an alternate user wants to access this file for writing, he includes the following control statement in his job.

```
ATTACH(XX7X/PW=ENTER,UN=AA29A,M=W)
```

If the originator of an indirect or direct access permanent file wants to change the access parameters for that file, he uses the CHANGE control statement. Format of this statement is as follows:

```
CHANGE(nfn=ofn/PW=passwor,CT=ct,M=m)
```

nfn — Optional new file name that will replace the old file name ofn. The keyword parameters PW=, CT=, and M= are included only if they are to be changed. If the password is to be cancelled, the user sets PW=0 (zero).

Example 4:

A direct access permanent file called ZZ3 has the password PASS2. Read permission has been granted to alternate users. The originator of this file changes the password to PASS3 and the access mode from read to write with the following control statement.

```
CHANGE(ZZ3/PW=PASS3,M=W)
```

HOW TO OBTAIN A LISTING OF PERMANENT FILES

The system maintains a separate catalog of each user's permanent files. A user's catalog contains the names, characteristics, and histories of his current permanent files. Catalogs are continually updated as permanent files are created, accessed, modified, or purged. A user can obtain a listing of permanent files (his and others he can access) by including the CATLIST control statement in one of his jobs. The listing is put in the OUTPUT file for that job.

A user can obtain a listing of the names of his permanent files with the following version of the control statement.

```
CATLIST.
```

A user can obtain a listing of the names of permanent files of an alternate user that he can access with the following version of the control statement.

```
CATLIST(UN=usernum)
```

usernum — User number of the alternate user

If the user wants the names, characteristics, and histories of all his permanent files, he uses the following control statement.

```
CATLIST(LO=F)
```

If the user wants the names, characteristics, and histories of all the permanent files in an alternate user's catalog that he can access, he uses the following control statement.

```
CATALOG OF usernum                          yy/mm/dd. hh.mm.ss.

   FILE NAME ACCESS FILE-TYPE LENGTH DN CREATION LAST ACCESS LAST MOD
    PASSWORD MD/CNT INDEX  PERM  SUBSYS DATE/TIME DATE/TIME DATE/TIME
```

Figure 5-2. CATLIST Parameter LO=F Heading

CATLIST(LO=F,UN=usernum)

usernum	User number of the alternate user

The headings printed out for a full listing (LO=F) are given in figure 5-2. The terms in these headings have the following significance.

yy/mm/dd	Year, month, and day of the listing.
hh.mm.ss	Listing time in hours, minutes, and seconds.
FILE NAME	Name of each file listed after a sequence number.
PASSWORD	File password if one is given. This term is missing from listings of alternate catalogs.
ACCESS MD/	This entry will be IND if the file is indirect access or DIR if the file is direct access.
/CNT	Count of the number of times the file has been accessed (originator and alternate users).
FILE-TYPE	Access category (PRIVATE, SEMI-PRIVATE, or PUBLIC).
INDEX	Reserved for system use.
PERM.	Permission mode. In the examples in this guide, this can be READ or WRITE.
LENGTH	File length given in decimal number of characters. The minimum that will be shown is 640 characters however small the file.
SUBSYS	Files created in a time-sharing session with a subsystem associated with them (FTNTS, BASIC, BATCH, or EXECUTE). If this field is blank, no subsystem is associated with the file.
DN	Gives a device number for direct access files. An asterisk in this column indicates the file is on the user's master device. This device holds the user's catalog and all his indirect access permanent files.
CREATION DATE/TIME	Two-line entry giving the date and time of file creation.
LAST ACCESS DATE/TIME	Two-line entry giving the date and time of the last access of this file.
LAST MOD DATE/TIME	Two-line entry giving the date and time of the last modification of this file.

Refer to the NOS Reference Manual, Volume 1 for additional catalog listings.

A user inserts control language statements into the control statement record of a job to give the sequence of control statements a program-like structure; that is, normal unconditional processing of control statements in series is modified so that tests, transfers, and loops are initiated within these statements as if they were lines in a program. This section outlines the following control language statements.

SET

DISPLAY

IF

GOTO

CALL

Only the basic parameters needed by the applications programmer are included. The full range of parameters is given in section 4 of the NOS Reference Manual, Volume 1.

FORMAT

A control language statement consists of a descriptive name followed by symbolic names and/or an expression. Separators and terminators are unique and must be used as shown in the individual formats.

EXPRESSIONS

The expressions used with control language statements are similar to the expressions used with higher-level languages. These may contain constants, arithmetic operators, relational operators, boolean operators, functions, and symbolic names (refer to appendix E for a listing of the operators).

FILE FUNCTION

The FILE function is used as a parameter in the SET, DISPLAY, and IF control language statements to determine the status of any file assigned to the job. Status includes file type, location, and accessibility. The format of the FILE function is as follows:

FILE(lfn,expression)

lfn	Name of the file for which status is being determined
expression	Legal expression; parentheses must be used

The special symbolic names used in this expression specify the status to be determined. Evaluation of the expression will give a value of 1 if it is true and a value of 0 if it is false. The following are symbolic names and their use.

File Type:

LO	Local
PR	Print
IN	Input
PH	Punch
PM	Direct access permanent file

File Location:

AS	File assigned to the user's job
MS	Mass storage
MT	7-track magnetic tape
NT	9-track magnetic tape

The FILE function is explained in subsequent descriptions and examples.

SET STATEMENT

SET control language statement sets a value in one of the software registers reserved for control language use. Format of the statement is as follows:

SET(Ri=expression)

i	Identifies one of three 18-bit software registers; can be 1, 2, or 3.
expression	Legal expression; parentheses must be used.

The boolean values TRUE or T and FALSE or F can be specified as the expression. These values are stored as 1 and 0, respectively.

As an example, the following SET statement included in a control statement loop will increase R1 by one each time the loop is processed.

SET(R1=R1+1)

Additional uses of the SET statement are given in the examples for other control language statements.

DISPLAY STATEMENT

The DISPLAY control language statement evaluates an expression and displays the result in the job's dayfile. Numerical values are displayed in decimal and octal. If the evaluation gives a true/false value, a 1 or 0 will be displayed, respectively. Format of the DISPLAY statement is as follows:

```
DISPLAY(expression)
```

expression	Legal expression; parentheses must be used.

Example 1:

An input copy operation is repeated in a control statement loop using:

```
SET(R1=R1+1)
```

A second copy operation is repeated with:

```
SET(R2=R2+1)
```

To obtain a printout of the total number of copies made, the user includes the following in the control statement record.

```
SET(R3=R1+R2)
DISPLAY(R3)
```

If R1 is 8 and R2 is 3, the entry in the dayfile is as follows:

```
DISPLAY(R3)
          11          13B
```

The user can employ the single statement

```
DISPLAY(R3=R1+R2)
```

and get the same entry in the dayfile.

Example 2:

The following statements make a local copy of a permanent file and test to see if it is on mass storage.

```
GET(ALPHA)
SET(R1=FILE(ALPHA,MS))
DISPLAY(R1)
```

If ALPHA is on mass storage, the following entry is made in the job's dayfile.

```
DISPLAY(R1)
          1           1B
```

If ALPHA is not on mass storage, the following entry is made in the job's dayfile.

```
DISPLAY(R1)
          0           0B
```

GOTO STATEMENT

The GOTO control language statement initiates an unconditional transfer of control statement processing to a named statement. Format of the statement is as follows.

```
GOTO,stmt.
```

stmt	Identification of the statement in the control statement record to which processing will transfer; the comma and period are required. The statement identification may be any of the following. • Name of a control statement (GET, COPYBR, and so forth). • Name of a control language statement (SET, DISPLAY, and so forth). • Alphanumeric identifier placed at the beginning of the statement to which control is to be transferred. The statement identifier must begin with a decimal digit and may have up to six alphanumeric characters following this digit. The identifier is separated from its statement by a comma.

If two or more statements in the control statement record have the same name, control will pass to the first occurrence.

Example 1:

In the following loop from a control statement record, one GOTO repeats the loop when the exit condition is not met. The other GOTO causes an exit from the loop when the exit condition is met.

```
    .
    .
2LOOP,DISPLAY(R1)
    .
    .
SET(R1=R1+1)
(exit condition)GOTO,COMMENT.
GOTO,2LOOP.
COMMENT.      END
```

Example 2:

Figure 6-1 shows a job demonstrating various possibilities of the GOTO statement; the resulting dayfile is shown in figure 6-2.

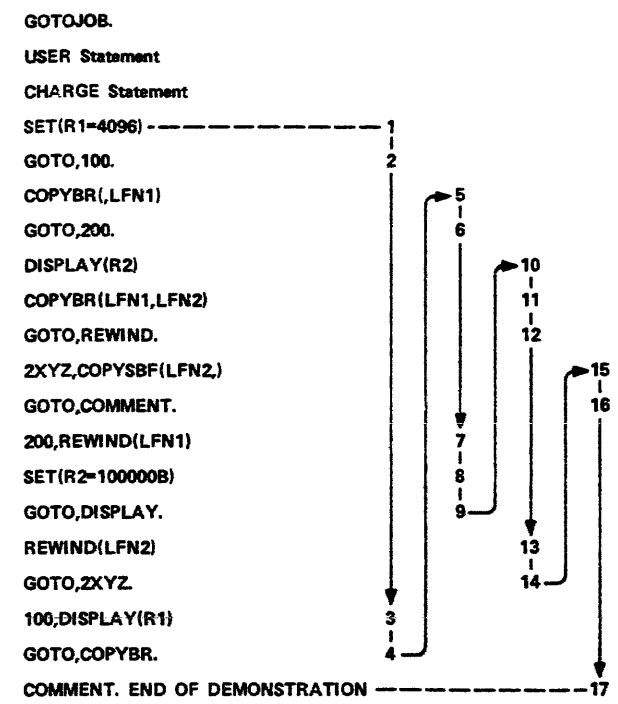


Figure 6-1. GOTO Job Demonstration

```
17.07.19.EFD.
17.07.19.USER(EFD2501,)
17.07.19.CHARGE(404,ACCT4)
17.07.20.SET(R1=4096)
17.07.20.GOTO,100.
17.07.21.100,DISPLAY(R1)
17.07.21.       4096       10000B
17.07.21.GOTO,COPYBR.
17.07.22.COPYBR(,LFN1)
17.07.23.  COPY COMPLETE.
17.07.24.GOTO,200.
17.07.24.200,REWIND(LFN1)
17.07.24.SET(R2=100000B)
17.07.24.GOTO,DISPLAY.
17.07.24.DISPLAY(R2)
17.07.24.      32768      100000B
17.07.25.COPYBR(LFN1,LFN2)
17.07.25.  COPY COMPLETE.
17.07.25.GOTO,REWIND.
17.07.25.REWIND(LFN2)
17.07.27.GOTO,2XYZ.
17.07.29.2XYZ,COPYSBF(LFN2,)
17.07.29.  END OF INFORMATION ENCOUNTERED.
17.07.29.GOTO,COMMENT.
17.07.29.COMMENT. END OF DEMONSTRATION****
17.07.30.UEAD,             0.002KUNS.
17.07.30.UEPF,             0.015KUNS.
17.07.30.UEMS,             0.0540KUNS.
17.07.20.UECP,             0.119SECS.
17.07.30.AESR,             2.205UNTS.
17.07.38, 32,                  0.320 KLNS.
```

Figure 6-2. Resulting Dayfile from GOTO

IF STATEMENT

The IF control language statement is analogous to the conditional branch found in higher-level languages. It states a condition which, if met, initiates processing of a statement on the same line; if the condition is not met, the statement is ignored and control passes to the next statement. Format of the IF statement is as follows:

IF(expression)stmt.

expression	Legal expression
stmt	Control statement or control language statement

The parentheses and period must be used as shown.

Example 1:

The following example tests the file TERM to see if it is a local file. If it is, control passes to the statement identified with 200; if it is not, register R1 is set to 0.

```
IF(FILE(TERM,LO)) GO TO , 200.
S E T (R1 = FALSE)
    .
    .
    2 0 0 , * T R U E
```

Example 2:

IF(R1=R2)DISPLAY(R1).

If these registers have the same value, that value will be displayed in the dayfile.

CALL STATEMENT

The CALL statement enables the user to insert preestablished procedures (procedure files) in the control statement record of a job. A procedure file consists of control statements and control language statements organized into a routine that is repeatedly used in jobs submitted by the user. Instead of re-creating this routine for successive jobs, the user inserts a call to the procedure file. When the CALL statement is processed, the procedure file is added after the CALL statement in the job's control statement record and becomes a part of that record. The basic format of the CALL statement is as follows:

CALL(lfn)

lfn	Name of the procedure file to be inserted in a control statement record. If a local file with the name lfn is not found, the system does a GET internally.

The name of the procedure file may appear as the first statement of the file; however, it is an optional reference and is not part of the insertion. This name may or may not have a terminator.

If the user wants control to be initially transferred to a statement in the procedure file other than the first, he uses the following format.

```
CALL(lfn,S=ccc)
```

ccc	Statement identifier of any statement in the procedure file (any one of the previously explained forms, refer to Statement Format)

Example 1:

The following procedure file is made into an indirect access permanent file.

```
DEMO
COPYBR(,TEMP)
REPLACE(TEMP=CONTROL)
GOTO,COMMENT.
GET(DEFER)
COPYSBF(DEFER,)
GOTO,COMMENT.
400,GET(CONTROL)
COPYSBF(CONTROL,)
COMMENT.        E N D
```

The first statement DEMO is the name of the procedure file and does not have a terminator.

A job's control statement record contains the following statement.

```
CALL(DEMO)
```

The following relevant portion of the dayfile results.

```
13.25.44.CALL(DEMO)
13.25.44.COPYBR(,TEMP)
13.25.45. COPY COMPLETE.
13.25.45.REPLACE(TEMP=CONTROL)
13.25.45.GOTO,COMMENT
13.25.45.COMMENT.       E N D
```

A job's control statement record contains the following statement.

```
CALL(DEMO,S=GET)
```

The following is the relevant portion of the dayfile that results.

```
09.11.55.CALL(DEMO,S=GET)
09.11.56.GET(DEFER)
09.11.56.COPYSBF(DEFER,)
09.11.56. END OF INFORMATION ENCOUNTERED.
09.11.57.GOTO,COMMENT.
09.11.57.COMMENT.       E N D
```

A job's control statement record contains the following statement.

```
CALL(DEMO,S=400)
```

The following is the relevant portion of the dayfile that results.

```
14.33.21.CALL(DEMO,S=400)
14.33.21.400,GET(CONTROL)
14.33.21.COPYSBF(CONTROL,)
14.33.22. END OF INFORMATION ENCOUNTER
14.33.22.COMMENT.       E N D
```

Example 2:

This example demonstrates the nesting of calls to procedure files. Two FORTRAN programs convert English units to metric units; the first program converts weights, and the second program converts lengths. The following job compiles these programs and saves the object code as indirect access permanent files.

```
METRICS.
USER Statement
CHARGE Statement
FTN(B=WEIG)
SAVE(WEIG)
FTN(B=LENG)
SAVE(LENG)
7/8/9
```

FORTRAN source program to convert weights

```
7/8/9
```

FORTRAN source program to convert lengths

```
6/7/8/9
```

The following procedure file makes a local copy of WEIG and executes it.

```
WEIGHT
GET(WEIG)
WEIG.
```

The following procedure file makes a local copy of LENG and executes it.

```
LENGTH
GET(LENG)
LENG.
```

These files are made indirect access permanent files with the names WEIGHT and LENGTH.

The following procedure file can call either WEIGHT or LENGTH.

```
METRIC
CALL(WEIGHT)
GOTO,COMMENT.
2LENG,CALL(LENGTH)
COMMENT.        **END**
```

If the control statement record calls this procedure file with

CALL(METRIC)

the procedure file METRIC will in turn call the procedure file WEIGHT which will initiate execution of the program WEIG. An input record of English weights will be converted to metric.

If the control statement record calls the procedure file METRIC with

CALL(METRIC,S=2LENG)

the program LENG will be executed, and an input record of English lengths will be converted to metric.

The expansion of the control statement record by these calls is diagrammed in figure 6-3. The dayfile resulting from CALL(METRIC) is shown in figure 6-4, and the dayfile resulting from CALL(METRIC,S=2LENG) is shown in figure 6-5.

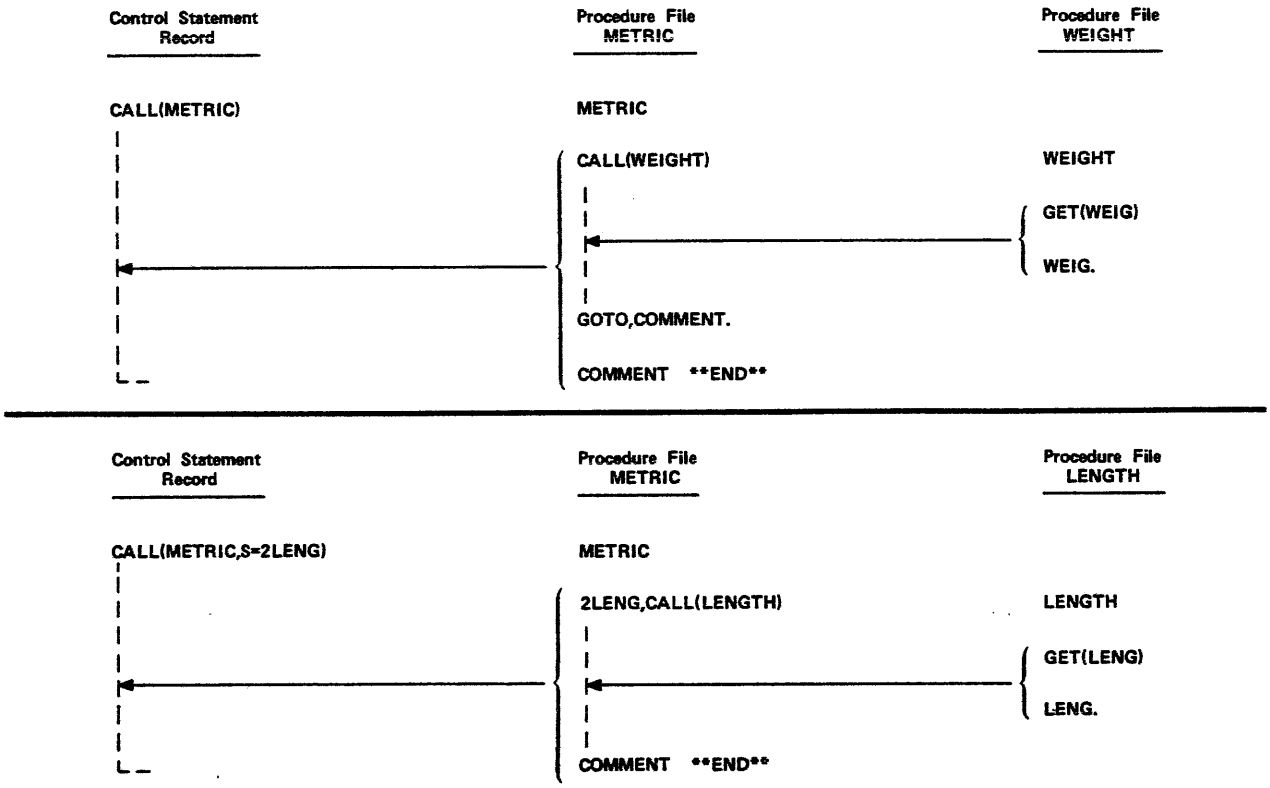


Figure 6-3. Control Statement Record Expansion with Nested Calls

```
17.32.34.METRIC1.
17.32.35.USER(JC8501,)
17.32.35.CHARGE(1010,NOS11)
17.32.36.CALL(METRIC)
17.32.36.CALL(WEIGHT)
17.32.37.GET(WEIG)
17.32.38.WEIG.
17.32.40.     STOP
17.32.40.        .006 CP SECONDS EXECUTION TIME
17.32.41.GOTO,COMMENT.
17.32.41.COMMENT. **END**
17.32.41.UEAD,       0.002KUNS.
17.32.41.UEPF,       0.025KUNS.
17.32.41.UEMS,       1.313KUNS.
17.32.41.UECP,       0.599SECS.
17.32.41.AESR,       3.107UNTS.
17.32.47.ULCP, 32,       0.128 KLNS.
```

Figure 6-4. CALL(METRIC) Dayfile

```
08.52.25.METRIC2.
08.52.25.USER(JC8501,)
08.52.25.CHARGE(1010,NOS11)
08.52.25.CALL(METRIC,S=2LENG)
08.52.26.2LENG,CALL(LENGTH)
08.52.27.GET(LENG)
08.52.27.LENG.
08.52.28.     STOP
08.52.28         .006 CP SECONDS EXECUTION TIME
08.52.28.COMMENT. **END**
08.52.29.UEAD,       0.002KUNS.
08.52.29.UEPF,       0.025KUNS.
08.52.29.UEMS,       1.252KUNS.
08.52.29.UECP,       0.563SECS.
08.52.29.AESR,        3.050UNTS.
08.52.35.UCLP, 35,       0.128 KLNS.
```

Figure 6-5. CALL(METRIC,S=2LENG) Dayfile

The system normally terminates job processing if an error is generated. However, the user can restrict or suspend this error exiting by inserting combinations of the following error control statements in the job's control statement record.

EXIT

NOEXIT

ONEXIT

The programmer inserts an EXIT statement as a reentry point in the control statement record, where processing can resume instead of terminating when an error is generated. If error processing is in effect and a control statement produces an error, the system sequentially searches the remainder of the control statement record for an EXIT statement. If it finds an EXIT, it resumes processing with the statement following the EXIT. If it fails to find an EXIT, it terminates the job. If no error occurs, processing terminates when the EXIT statement is encountered (figure 7-1).

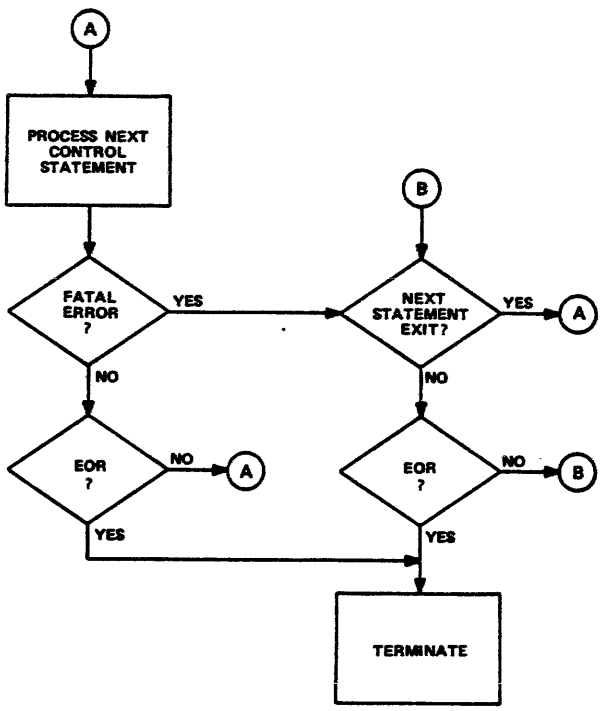


Figure 7-1. EXIT Statement Operation

If the user inserts a NOEXIT statement in the job's control statement record, error processing is suspended for the statements that follow. Error processing will remain suspended until the end of the job or until the system encounters an ONEXIT statement. With error processing suspended, the system will attempt to process each succeeding control statement no matter how many fatal errors are encountered.

Example 1:

The following control statements are included in the control statement record of a job.

```
COPYBR(INPUT,INFILE)
APPEND(BETA,INFILE)
GET(BETA)
COPYSBF(BETA,OUTPUT)
```

This adds an input record to an existing indirect access permanent file. A local copy is made of the extended permanent file and printed out. The relevant dayfile entries are as follows:

```
13.53.40.COPYBR(INPUT,INFILE)
13.53.40. COPY COMPLETE.
13.53.40.APPEND(BETA,INFILE)
13.53.41.GET(BETA)
13.53.42.COPYSBF(BETA,OUTPUT)
13.53.42. END OF INFORMATION ENCOUNTERED.
```

Example 2:

If the user has erroneously punched the first copy statement so that it reads

```
COPYB(INPUT,INFILE)
```

the dayfile would read as follows:

```
13.59.30.COPYB(INPUT,INFILE)
13.59.30. ILLEGAL CONTROL CARD.
```

The job terminates at this point.

Example 3:

If, in the previous example, an EXIT has been placed after the GET

```
COPYB(INPUT,INFILE)
APPEND(BETA,INFILE)
GET(BETA)
EXIT.
COPYSBF(BETA,OUTPUT)
```

the dayfile would read as follows:

```
14.02.00.COPYB(INPUT,INFILE)
14.02.00. ILLEGAL CONTROL CARD.
14.02.00.EXIT.
14.02.00.COPYSBF(BETA,OUTPUT)
14.02.01. END OF INFORMATION ENCOUNTERED.
```

As soon as the system encountered the illegal statement, it skipped to the EXIT statement and resumed processing.

Example 4:

If instead of the EXIT, in the preceding example, a NOEXIT had been placed ahead of all of these statements

```
NOEXIT.
COPYB(INPUT,INFILE)
APPEND(BETA,INFILE)
GET(BETA)
COPYSBF(BETA,OUTPUT)
```

the dayfile would read as follows:

```
14.05.44.COPYB(INPUT,INFILE)
14.05.44. ILLEGAL CONTROL CARD.
14.05.45.APPEND(BETA,INFILE)
14.05.45. INFILE NOT FOUND, AT 000121.
14.05.45.GET(BETA)
14.05.45.COPYSBF(BETA,OUTPUT)
14.05.45. END OF INFORMATION ENCOUNTERED.
```

Although the first copy statement is illegal, the system attempts to process each succeeding control statement.

The address in the job where the system looked for the ALPHA reference was 000121. This is important to the batch user when he begins taking dumps (refer to section 13, NOS Reference Manual, Volume 1), since they give the machine language representation of his job.

Example 5:

In the following example, two control statements have errors (COPYB instead of COPYBR and BATA instead of BETA in the GET). A NOEXIT precedes all the statements, and an ONEXIT follows the APPEND.

```
NOEXIT.
COPYB(INPUT,INFILE)
APPEND(BETA,INFILE)
ONEXIT.
GET(BATA)
COPYSBF(BETA,OUTPUT)
```

The relevant portion of the resultant dayfile is as follows:

```
14.17.16.NOEXIT.
14.17.16.COPYB(INPUT,INFILE)
14.17.16. ILLEGAL CONTROL CARD.
14.17.16.APPEND(BETA,INFILE)
14.17.16. INFILE NOT FOUND, AT 000121.
14.17.16.ONEXIT.
14.17.16.GET(BATA)
14.17.16. BATA NOT FOUND, AT 000121.
```

After NOEXIT, error processing is suspended, and even though the COPYB is illegal, the system tries to process the APPEND statement. After ONEXIT, error processing is again in effect, and the failure to find a permanent file called BATA ends job processing.

This section describes the use of labeled, magnetic tape files. The following control statements are included.

LABEL

RESOURC

The following descriptions and examples introduce magnetic tapes. The full range of tape capabilities under NOS is given in the NOS Reference Manual, Volume 1.

DEFINITIONS

The programmer who wants to use magnetic tapes should be familiar with the identification and layout of data on tapes (figure 8-1). The following paragraphs give fundamental definitions of these features.

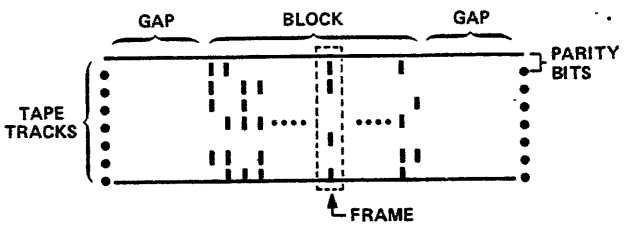


Figure 8-1. Magnetic Tape Data Layout

All the parameters which identify and prescribe the layout of data on magnetic tapes have defaults. Some of these defaults are standard with NOS; others are specified by the installation. Defaults are ignored in this section to familiarize the beginning tape user with the fundamental parameters.

TAPE TRACKS

Data can be written on magnetic tape in 7 or 9 parallel data paths (tracks), extending lengthwise on the tape. A particular model of a tape drive will read and write either 7-track exclusively or 9-track exclusively; there is no interchangeability between the two track types on a single drive.

The user specifies 7-track with the parameter MT and 9-track with the parameter NT. The examples in this section use MT.

DENSITY

The density of data written on a magnetic tape is the number of characters per lengthwise inch. This is measured in bits per inch (bpi) or characters per inch (cpi). The user specifies density with the parameter D=den. The values for den are the following.

den	Value	Track Type
LO	200 bpi	7
HI	556 bpi	7
HY	800 bpi	7
HD	800 cpi	9
PE	1600 cpi	9

The examples in this section use the specification D=HY.

RECORDING MODE

Records are written on tape in either binary or coded mode. Binary mode is the format used to store data in central memory. This is the 6-bit display code listed in appendix D. Binary mode can be used to record any data, and since no code conversion takes place, it is usually more efficient. Coded mode includes any of the formats used to store data on peripheral devices. For 7-track tapes, this can be 6-character external BCD or the 6-character subset of ASCII (refer to appendix D). For 9-track tapes, this can be 8-character ASCII or 8-character EBCDIC (refer to appendix A of the NOS Reference Manual, Volume 1). Coded mode writes character data (program source code, text, card input, information to be printed, and so forth); it cannot be used for binary data.

The user of this guide specifies recording mode by his selection of copy statements (refer to section 4).

The I format for tapes, described in this section, records in binary mode. Accordingly, the user can employ this format for writing and reading any data.

PARITY

Every tape is written with one track composed of check bits called parity bits. The system adds these bits during the write operation so that each frame will have an even number of bits (even parity) or an odd number of bits (odd parity). The remaining tracks (six for a 7-track tape and eight for a 9-track tape) will contain user data. When the tape is read, each frame is checked for proper parity. If any frame fails the test, an error message is issued to the dayfile, and the system rereads the block in which the error occurred. This reread is repeated a number of

times. If the error was transient and the frame passes the parity test, reading continues; otherwise, reading terminates.

A 7-track tape uses even parity for coded records and odd parity for binary records. A 9-track tape uses only odd parity.

Ordinarily, the user has no control over parity but should understand its significance to interpret tape error messages.

BLOCKS

A tape drive needs a short length of tape for stop-and-go operations. Hence, the drive cannot position to individual characters but must transfer data in parcels to and from memory where the individual data items can be addressed. These parcels are called blocks. Blocks are spaced on tape with the necessary blank gaps between them. Block length or size is specified in characters/block. The length of a block may be fixed or variable, depending upon the format used.

VOLUME

In tape terminology, a volume is the set of all files on one reel of tape. This may be an empty set, or the set may occupy all or part of the reel. There can be but one volume per reel; hence, the term is often used as a synonym for a reel of tape.

VOLUME SERIAL NUMBER

A volume serial number (VSN) is one to six alphanumeric characters that identify the set of files on a single reel of tape. A VSN is handwritten on the outside of a reel (external label) and, for labeled tapes, entered in the first internal label of the tape. If an installation is issuing tapes to users, it will specify VSNs. These installation tapes will have the VSN on an external label and, if labeled tapes are being used, they will be blank-labeled. A tape is blank-labeled when the format of the first internal label is written on the tape with all entries except the VSN blank. If the user has his own tapes and is specifying his own VSNs, he will handwrite his VSNs on the outside of the reels he submits with his jobs. If the tape is to be labeled, he will request the operator to blank-label it with the VSN specified. Customarily, this request is on a form that accompanies the job. Some installations permit users to blank-label their tapes with the BLANK control statement (refer to the NOS Reference Manual, Volume 1).

The system uses the VSN on the internal label of a labeled tape to assign that reel to a user's job via the lfn the user specifies on a LABEL control statement. During the remainder of job processing, control statements that contain this lfn access the reel of tape with the associated VSN. The lfn is released when the job terminates; the VSN remains with the reel until specifically changed.

TAPE LABELS

A NOS-labeled tape uses internal tape labels for identification and indexing. Each label consists of 80 characters. The first label is the volume header label which is written at the beginning of the tape. It contains the VSN and owner identification (user number/family name) for the tape. Following this, every file copied to the tape will have a file header label before the first block of data and an EOF label after the last block of data. These file delimiting labels can contain indexing parameters to access individual files by name and/or sequence number. Indexing of multifile tapes is explained in the NOS Reference Manual, Volume 1.

TAPE FORMAT

The format of a magnetic tape is the physical layout of data on the tape. This can include block size, mode (binary/coded), internal tape labels, record and file delimiters, and special features of data definition. Eight formats are presently available under NOS (refer to section 10 of the NOS Reference Manual, Volume 1). Any one format is specified on a tape control statement with the keyword F=format; the examples in this section use I format. This format is the most reliable since, on read operations, the system checks the number of bytes read with the number expected and uses this check to detect incomplete or missing blocks.

HOW TO CREATE A LABELED TAPE

The user creates a labeled tape with the following basic version of the LABEL statement.

```
LABEL(lfn,VSN=vsn,D=den,tr,F=format,W)
```

lfn	Local file name temporarily given to the data on the tape (whether it is recorded or is about to be written). This parameter must come first; the others are order-independent.
vsn	Identifies the physical reel of tape. It is used to assign the reel of tape to the job.
den	Tape density (LO, HI, HY, HD, or PE).
tr	Track type (MT or NT).
format	Tape format in which data will be written on the tape (I, SI, X, S, L, E, B, or F).
W	Specifies the writing of internal file identifying labels by the system.

When the LABEL statement is processed, the system determines which tape drive has a tape mounted with the specified vsn and automatically assigns that tape to the job via the lfn in the LABEL statement.

Example:

A user has a reel of tape that has been blank-labeled with VSN=MAG5. He wants to copy a file from his input deck to this tape. He includes the following statements in the deck.

```
LABEL(TFILE,VSN=MAG5,D=HY,MT,F=I,W)
COPYBR(,TFILE)
```

When job processing encounters the LABEL statement, the system automatically assigns the reel of tape identified with VSN=MAG5 to this job via the name TFILE. Data written on this tape will be recorded at a density of 800 bpi on a 7-track unit. Since I format is specified, the data will be written in binary mode. Binary records on 7-track have odd parity.

The COPYBR statement copies an input record to the tape. Subsequent control statements in the same job can access this tape record with the name TFILE. These job steps are recorded in the job's dayfile as follows:

```
LABEL(TFILE,VSN=MAG5,D=HY,MT,F=I,W)
MT53, ASSIGNED TO TFILE     , VSN=MAG5    .
COPYBR(,TFILE)
  COPY COMPLETE.
```

MT53 specifies that the tape is mounted on the magnetic tape drive with equipment number 53.

Later in the same job, this file is printed with the following statements.

```
REWIND(TFILE)
COPYSBF(TFILE,)
```

HOW TO ACCESS A LABELED TAPE

The user can access an existing labeled tape with the following format of the LABEL statement.

```
LABEL(lfn,VSN=vsn,D=den,tr,F=format,R)
```

R	Specifies read-label which initiates a system comparison of the present values in the internal tape labels with the parameters on the LABEL statement. If the comparison fails, the job aborts.

After processing this statement, the tape is attached to the job via the local file name lfn. Subsequent control statements in the same job can read or write information on this tape.

If a W is included in the LABEL statement instead of an R, this will create a new labeled tape, and all the existing data will be destroyed.

Before anything can be written on a reel of tape, a write ring must be inserted. The hardware requires an explicit action (insertion of the write ring) before it will record data on a tape to guarantee that existing data will not be accidentally written over. To ensure that this ring is inserted, the user may include the optional parameter PO=W. If the PO=W parameter is included and the tape is mounted without the write ring, job processing will be suspended until the operator remounts the tape with a ring. If W is specified, PO=W is not needed to enforce ring in.

The user can ensure that the write ring is left out so the tape is not accidentally written on by including the parameter PO=R. If this parameter is included and the tape is mounted with a write ring inserted, job processing is suspended until the operator removes the ring.

Example:

To access the tape file created in the previous example and add information from the INPUT file, the following control statements are included in a subsequent job.

```
LABEL(DEMO2,VSN=MAG5,D=HY,MT,F=I,PC=W,R)
SKIPEI(DEMO2)
COPYBF(,DEMO2)
```

The tape file is given a new lfn (DEMO2), but the vsn for the reel (MAG5) remains the same. DEMO2 refers to the data on this reel, and MAG5 refers to the physical reel of tape.

If the SKIPEI statement had not been included, the input record would have been copied over the old information.

HOW TO COPY FROM ONE TAPE TO ANOTHER

The user can copy one tape to another by including two LABEL statements (one for each tape) in the job. However, a RESOURC statement must come before the second or both LABEL statements. The RESOURC statement is required whenever two or more tape drives will be used concurrently. The format of this statement is as follows:

```
RESOURC(rt=u)
```

rt	MT for 7-track or NT for 9-track
u	Number of tape drives to be used by the job concurrently

Example:

A job uses the following control statements to copy one tape to another and verify the copy.

```
RESOURC(MT=2)
LABEL(DEMO3,VSN=MAGT1,D=HY,MT,F=I,R,PO=R)
LABEL(DEMO4,VSN=MAGT2,D=HY,MT,F=I,W)
COPYEI(DEMO3,DEMO4)
VERIFY(DEMO3,DEMO4,R,N=0)
```

The RESOURC statement specifies two 7-track tape drives for the job. The first LABEL statement assigns the source of the copy to one drive, the R specifies the checking of labels, and PO=R ensures that DEMO3 is used only for reading and is not written on. The second LABEL statement assigns the reel of tape to receive the copy, and W specifies the writing of labels. After the copy is made, it is verified. The dayfile entries for this sequence are as follows:

```
RESOURC(MT=2)
LABEL(DEMO3,VSN=MAGT1,D=HY,MT,F=I,R)
MT53,  ASSIGNED TO DEMO3     , VSN=MAGT1  .
LABEL(DEMO4,VSN=MAGT2,D=HY,MT,F=I.W)
MT54,  ASSIGNED TO DEMO4     , VSN=MAGT2  .
COPYEI(DEMO3,DEMO4)
 COPY COMPLETE.
VERIFY(DEMO3,DEMO4,R,N=0)
 VERIFY GOOD.
```

TAPE ERROR MESSAGES

Tape errors result from physical causes (which may be transient) or incorrect formats (label missing, improper block length, and so forth). When NOS encounters a tape-related error, it issues a 3- or 4-line message to the job's dayfile. The following message is typical.

```
MT,C13,03,MT2     ,RD,53,SO,GS4203.
MT,C13,D10000100020000000502000000000120
MT,C13,FO7,I00,B000000,L0050,P00000000.
MT,C13,E30,H4063,540, STATUS.
```

The following entries are of interest to the batch user.

First line:

MT2	Volume serial number of the reel of tape.
RD	Read operation; WD indicates a write operation.
53	Equipment number of the tape drive.

Fourth line:

STATUS	Indicates that recovery was not possible with this attempt. If in any of these attempts, recovery is successful, STATUS will be replaced with RECOVERED. If recovery fails after a number of attempts, STATUS is replaced with the error description. Typical entries are WRONG PARITY and BLOCK SEQUENCE ERROR (refer to the NOS Reference Manual, Volume 1).

This section describes deferred batch processing in which a user types the line images of a batch job at a time-sharing terminal and submits them for processing as a batch job. This section also describes conversational batch processing in which the user enters control statements from a time-sharing terminal to be processed one at a time in the same manner as time-sharing commands. The following control statements are included in the descriptions.

SUBMIT

ENQUIRE

DAYFILE

ROUTE

This section assumes the user is familiar with terminal operation. Such terms as line number and primary file are used without definition. The mechanics of terminal usage are explained in the NOS Time-Sharing User's Guide and NOS Time-Sharing User's Reference Manual.

NOTE

In this section, user entries at the terminal are indicated with lowercase letters, and system responses are indicated with uppercase letters.

DEFERRED BATCH

The user may create a deferred batch job during a time-sharing session at a terminal. He types the job statements with reformatting directives (refer to Reformatting Directives), as required. If input records are to be used, they are typed after the statements. This sequence of line images and embedded reformatting directives constitute the submit file. The submit file enters processing when the user types the SUBMIT statement at the terminal.

Output from job processing can go to a printer, it can be made a permanent file which may be displayed at the terminal after job processing, or it can be dropped. Output may be dropped when the user is merely checking the dayfile to see how the job ran. When it runs satisfactorily, output can be initiated (refer to DAYFILE Statement).

The terminal user may include line numbers to make corrections before submitting the job; however, the submit file can be constructed under text mode (refer to the Time-Sharing Reference Manual) or with the Text Editor (refer to the NOS Text Editor Reference Manual).

The structure of the submit file is explained under Reformatting Directives. Entering the submit file for processing is explained under SUBMIT Statement. Monitoring deferred batch processing is outlined under ENQUIRE Statement and DAYFILE Statement.

REFORMATTING DIRECTIVES

The reformatting directives described in this guide are essential for establishing the limits and divisions of a batch job, inserting existing files, and deleting or retaining line numbers. Additional directives are described in the NOS Reference Manual, Volume 1.

All reformatting directives begin with a / to distinguish them from control statements; they do not have a terminator.

The terminal user signals the beginning of a deferred batch job by typing:

```
/JOB
```

He follows this with the job statement, USER statement, and CHARGE statement (if required). Typically, the first four statements are as follows:

```
/JOB
job statement
USER statement
CHARGE statement
```

Since multipunching is impossible on a terminal keyboard, special directives are available to signal the EOR and the EOF. The user specifies an EOR and EOF by typing:

```
/EOR
```

```
/EOF
```

An EOI directive is not needed with a deferred batch job.

The system will remove line numbers by default when it begins processing a deferred batch job. However, the user can retain line numbers for specific segments of the file by inserting the following directive.

```
/NOSEQ
```

The lines that follow this directive will retain their line numbers. If later in the file the user wants to return to the default, he inserts the following directive.

```
/SEQ
```

The system will remove the line numbers from all lines that follow this directive.

The user can insert an existing file in the sequence of lines that make up a submit file with the following directive.

/READ,lfn

The parameter lfn can be the name of a local file or a permanent file. If it is a permanent file, the system will automatically perform a GET or ATTACH.

Figure 9-1 shows the reformatting of a submit file. The left side of the figure shows the file as it is typed at the terminal; the right side shows the reformatted file that enters processing.

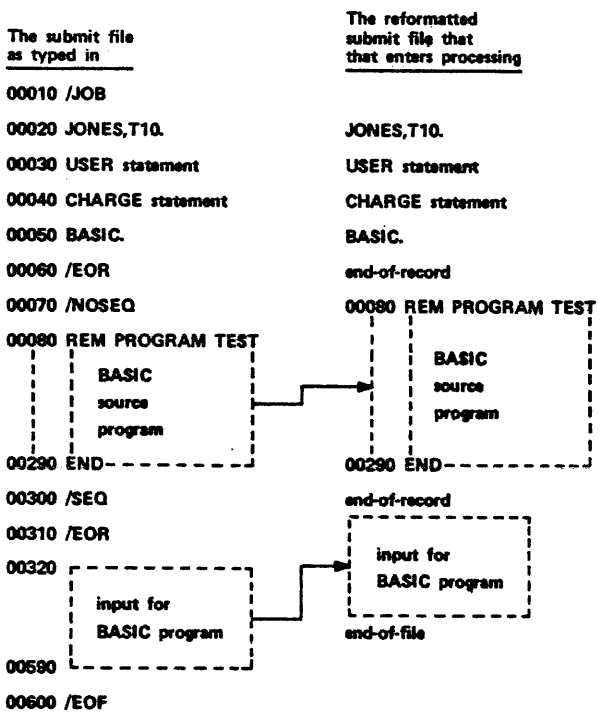


Figure 9-1. Reformatting a Submit File

SUBMIT STATEMENT

The time-sharing user enters the SUBMIT statement to initiate batch processing of a submit file he has created. The basic format of the statement is as follows:

SUBMIT,lfn,q

No terminator is needed, but the parameters are order-dependent. The lfn parameter is the name of the submit file. By default, lfn is the primary file. The q parameter specifies the disposition of output as follows:

B	Output goes to the central site printer.
N	Output is dropped at job termination, default from a time-sharing terminal.
E	Output goes to a remote batch terminal.

Example:

The following lines constitute a submit file the user creates and submits from a terminal. User entries are lowercase, and system responses are uppercase.

```
new,demo

READY.
auto
00100    /job
00110    myjob.
00120    user(efd25,ed)
00130    charge(58n2)
00140    copybf(,newlist)
00150    replace(newlist)
00160    /eor
00170    item one
00180    item two
  .
  .
00320    item last
00330    /eof
00340    *DEL*
submit
```

The abbreviated form of the SUBMIT statement implies two defaults, the submit file is the primary file (DEMO) and output will be dropped (N). Any one of the following versions of the statement accomplishes the same thing.

```
submit,demo,n

submit,,n

submit,demo

submit
```

As soon as the SUBMIT statement is processed, the system prints the time and the 7-character job name. A typical response is as follows:

```
 11.12.01. AKQIFIA
READY.
```

ENQUIRE STATEMENT

After a user has entered a submit file for processing, he can track the progress of the job with the following ENQUIRE statement.

```
ENQUIRE,JN=job
```

job	Last three characters of the name the system displays as soon as the SUBMIT statement is processed; no terminator is needed

The following system responses are possible.

```
jobname IN INPUT QUEUE

jobname EXECUTING

jobname IN ROLLOUT QUEUE

jobname IN PRINT QUEUE

jobname NOT FOUND
```

The last response indicates that processing is complete. The jobname parameter is the 7-character job designation.

After creating a submit file and entering it for processing with

```
submit,,b
```

the system responds as follows:

```
 17.15.19. AKQIBIN
READY.
```

The user then types:

```
enquire,jn=bin
```

or

```
enquire,jn=akqibin
```

The system responds as follows:

```
AKQIBIN IN PRINT QUEUE.
```

The user waits a moment and then types:

```
enquire,jn=bin
```

The system responds as follows:

```
AKQIBIN NOT FOUND.
```

The user now knows his job has completed processing and can pick up his printout.

DAYFILE STATEMENT

The time-sharing user can insert the DAYFILE statement into a submit file and obtain a copy of most of the dayfile resulting from processing. This enables the user to rerun and troubleshoot a deferred batch job without printouts. The DAYFILE statement is used in conjunction with the REPLACE statement as follows:

```
DAYFILE(lfn)
REPLACE(lfn)
```

lfn	User-supplied local file name for the dayfile

If these statements are included in a submit file, a copy of the resulting dayfile from the beginning (job statement) to the DAYFILE statement will be made a permanent file with the name lfn. The user can obtain a listing of this copy with the following statements.

```
GET,lfn
LNH,F=lfn
```

or

```
OLD,lfn
LNH
```

The following submit file obtains a copy of a compiled COBOL program that indexes the random name list included with the file.

```
00100   /job
00110   efd.
00120   user(efd25,d)
00130   charge,8823,693N55.
00140   get(alfa)
00150   alfa.
00160   dayfile(day)
00170   replace(day)
00180   /eor
00190   smith,a.j.
00200   green,l.e.
  .
  .
00340   jones,r.t.
00350   /eof
SUBMIT

  13.53.39.AKQIBGL
READY.
```

The user verifies that processing is complete and obtains a dayfile listing as follows:

```
get,day
lnh,f=day

13.06.08.EFD.
13.06.08.USER(EFD25,)
13.06.08.CHARGE,8823,693N55.
13.06.09.GET(ALFA)
13.06.09.ALFA.
13.06.14.SOURCE LINE 28
13.06.14.S T A R T  C O B O L  S O R T
13.06.14.
13.06.14.
13.06.15.93 RECORDS RELEASED
13.06.15.93 RECORDS RETURNED
13.06.15.E N D  C O B O L  S O R T
13.06.15.DAYFILE(DAY)
```

The user can make the saving of the dayfile dependent upon error processing by adding the following statements at the end of the control statement record.

```
DAYFILE(lfn)
REPLACE(lfn)
EXIT.
DAYFILE(lfn)
REPLACE(lfn)
```

The first two statements provide a dayfile if no error was detected. In that case, the EXIT causes job termination. The last two statements will be processed only if an error occurs. In that case, any error in the previous statements will initiate a skip to EXIT and the processing of the remaining statements.

LISTING BATCH OUTPUT AT A TERMINAL

The output from processing a deferred batch job can be listed at a terminal rather than printed by including the following statement in the control statement record of the submit file.

```
REPLACE(OUTPUT=pfn)
```

pfn	User-given name of the permanent file that is a copy of the job's OUTPUT file

The dayfile will not be included. If the user wants the dayfile, he should include a DAYFILE statement ahead of the REPLACE statement.

Example:

The following submit file compiles and executes a BASIC program which obtains its input from the submit file.

```
00100  /job
00110  smith.
00120  user(s221,sm)
00130  charge(923,92n1)
00140  basic.
00150  replace(output=display)
00160  /eor
00170  /noseq
00180  rem basic program
00190  input b1,b2,b3
00200  print b3,b2,b1
00210  end
00220  /seq
00230  /eor
00240  11111,22222,33333
00250  /eof
```

After the user verifies that processing is complete, he types:

```
get,display
lnh,f=display
```

He then receives the following listing at the terminal.

```
1       BASICXX                              BASIC 3.1
76316   77/01/27. 15.28.50.   PAGE   1
0
        00190 REM BASIC PROGRAM
        00200 INPUT B1,B2,B3
        00210 PRINT B3,B2,B1
        00220 END


33333        22222        11111
```

CONVERSATIONAL BATCH

Conversational BATCH refers to the batch subsystem, one of the subsystems available under time-sharing operation. In the batch subsystem, the user can enter batch control statements and time-sharing commands. Like a time-sharing command, each batch statement entered is immediately processed.

The user enters the batch subsystem by typing:

```
batch
```

The system responds as follows:

```
$RFL,0.
/
```

RFL refers to the running field length for the entries that follow. Zero indicates that the system will determine the field length for each entry. In most cases, the user need not change this (refer to the NOS Reference Manual, Volume 1).

The user then types a batch statement or time-sharing command after the slash. A terminator is not required except with control language statements. When the user presses carriage return, this entry is processed. If the processing produces a message or listing, this follows the entry. A new slash will appear, and the user can type another statement or command. The user exits from the batch subsystem by typing the name of another time-sharing subsystem (NULL, FTNTS, BASIC, EXECUTE).

In conversational batch, the time-sharing user is no longer restricted to time-sharing commands and can use batch control statements such as the copy statements, file positioning statements, and the ROUTE statement. The ROUTE statement has the following format.

ROUTE(lfn,p_1,p_2,...p_n)

lfn Name of a file to be routed to some device.

p_i Specifications of destination, data format, and print forms (refer to the NOS Reference Manual, Volume 1). The following options are frequently used.

p_i		Disposition Code
DC=xx	xx	
	LP	Print on any printer
	SB	Punch system binary
	PU	Punch coded

p_i		External Characteristics
EC=xx	xx	
	ASCII	Punch ASCII
	026	Punch 026 mode
	029	Punch 029 mode

Example:

The following entries in a time-sharing session obtain a local copy of a permanent file called INVTORY, make a copy of the 23rd record, and route that record to any printer. The user then picks up the printout of the record.

```
new,text

READY.
batch
$RFL,0.
/get,invtory
/skipr,22
SKIPR,22.
/copybr,invtory,print1
 COPY COMPLETE.
/rewind,print1
$REWIND,PRINT1.
/route,print1,dc=lp
 ROUTE COMPLETE.
/dayfile
 15.16.59.$CHARGE,966,N24.
 15.17.24.OLD,INVTORY.
 15.18.21.NEW,TEST.
 15.18.27.$RFL,0.
 15.18.56.GET,INVTORY.
 15.19.05.SKIPR,22.
 15.19.43.COPYBR,INVTORY,PRINT1.
 15.19.44.  COPY COMPLETE.
 15.19.57.$REWIND,PRINT1.
 15.20.11.ROUTE,PRINT1,DC=LP.
 15.20.14.  ROUTE COMPLETE.
 15.20.29.$DAYFILE.
 USER DAYFILE DUMPED.
```

SKIPR, COPYBR, and ROUTE batch control statements are available to the time-sharing user only under conversational batch. The printout initiated with ROUTE does not have a dayfile. If the time-sharing DAYFILE command is included, it gives the sequence of command and control statement processing from the CHARGE to the DAYFILE statement.

ADDRESS	The location of a word in memory. The location is designated by number or symbolic name.
BASIC	The Beginner's All-purpose Symbolic Instruction Code, a higher-level language originally designed and implemented at the Dartmouth College Computation Center. Programs written in a higher-level language have to be translated (compiled) into object code (machine language) before they can be executed.
BATCH JOB	A sequence of control statements and optional programs and input data that are submitted as a self-contained unit and processed without user intervention.
CATALOG	Each user has a catalog of his permanent files that is maintained by the system. Whenever he creates, modifies, or purges one of his permanent files, the system updates his catalog to reflect that action.
CHARGE NUMBER	An alphanumeric identifier the installation uses to allocate charges to individual users for system usage.
COBOL	COmmon Business Oriented Language. This higher-level language simplifies the programming of business data applications. The format of programs written in this language is similar to simple English sentences. These programs must be translated (compiled) into machine language before they can be executed.
COMPILER	A system product that translates source code programs into object code; that is, machine language (compilers are available for FORTRAN, COBOL, and BASIC).
CONTROL LANGUAGE	A set of statements which the user can insert into the control statement record of a job to initiate skips, conditional transfers, and iterations in the processing of the statements. It also affords the capability of setting and displaying software registers, as well as calling procedure files.
CONTROL STATEMENT RECORD	The first record of every batch job. This record contains all the control statements that specify the tasks that the system is to perform.
CONVERSATIONAL BATCH	The BATCH subsystem available under time-sharing operation. This subsystem allows the time-sharing user to enter batch control statements, as well as time-sharing commands, and obtain an immediate response to each entry. This distinguishes it from the other forms of batch in which statements are submitted and processed as a group.
DEFAULT	A value supplied by the system when the user omits its specification from a parameter list on a control statement.
DEFERRED BATCH	A batch job typed at a time-sharing terminal or included within another batch job and submitted for processing as a self-contained unit.
DIRECT ACCESS FILE	A permanent file on mass storage that is accessed without an intermediate local copy being made.
FAMILY NAME	A designation that the installation may give to a group of permanent file devices.
FILE	A contiguous collection of information that is accessed by name. It is delimited by a BOI and an EOI. It may be further divided into records.
FORTRAN	FORmula TRANslation, a higher-level language consisting of symbols and statements that can be used to create a program closely resembling mathematical notation. This program must be translated (compiled) into machine language before it can be executed.
HIGHER-LEVEL LANGUAGE	A programming language written with a defined set of mnemonic words and translated (compiled) into machine language for execution (for example, BASIC, COBOL, and FORTRAN).
INDIRECT-ACCESS FILE	A permanent mass storage file that can be accessed only by making a copy that is local to a particular job. When the job terminates, the copy is dropped, but the original remains.
INPUT FILE	The system-defined file which contains the entire job the user submits for processing. It is also known as the job file.
JOB FILE	Refer to INPUT FILE.

JOB STATEMENT	The first control statement of a batch job. It identifies the job and may specify a time limit and/or a memory limit.
LOCAL	Refers to data that exists only during the processing of a single job and that can only be accessed by that job.
LOCAL BATCH	A batch job submitted at the central computer site.
LOCAL FILE	A file copy created during the processing of a single job by a copy statement, program, or GET. When the job terminates, the copy is dropped.
OBJECT CODE	The machine language version of a program that has been translated (compiled) from source code written in an original higher-level language.
ORDER-DEPENDENT	Parameters in a control statement that must appear in a specified order.
ORDER-INDEPENDENT	Parameters in a control statement that may appear in any order, because they are specified with keywords.
OUTPUT FILE	The system-defined file which contains all the output from job processing. It is also known as the print file.
PARITY	In writing data, an extra bit is either set or cleared in each byte so that every byte has either an odd number of set bits (odd parity) or an even number of set bits (even parity). Parity is checked on a read for error detection and possible recovery.
PASSWORD	A password is either an alphanumeric value the user may have added to his user number to increase the security of the user number or an alphanumeric value the user may add to one of his permanent files with the PW parameter to increase the security of that file.
PHYSICAL RECORD UNIT (PRU)	The physical structure of a file as determined by the storage medium. For mass storage, a PRU is 64 central memory words (640 characters); for magnetic tape, the size of the PRU depends upon the tape format.
PERMANENT FILE	A file retained on mass storage until it is specifically purged.
PRINT FILE	An OUTPUT file containing data to be printed at a central site line printer.
PROJECT NUMBER	An alphanumeric identifier the installation uses to allocate charges for system usage to individual projects.
RECORD, LOGICAL	A user-defined subdivision of a file.
REMOTE BATCH JOB	A job submitted from a remote batch terminal.
SEPARATOR	A character used to separate parameters in a control statement.
SOURCE CODE	A program written in a higher-level language (FORTRAN, BASIC, or COBOL) which must be translated (compiled) into object code (machine language) before it can be processed.
SYSTEM RESOURCE UNIT (SRU)	An accounting unit that is a composite of central processor time, I/O activity, and memory used.
TERMINATOR	The character that must end every control statement included in a local batch, remote batch, or deferred batch job.
USER NUMBER	An alphanumeric identifier assigned to a user which uniquely identifies that user. The user must specify this identifier on his USER statement before he can gain access to the system.

The system appends a history of control statement processing to the OUTPUT file of every job. This is called the dayfile. It begins with a heading that gives the 7-character job name, the date, and the installation identification. This heading is followed by a list of all control statements processed by the job. Comments and diagnostics are added where applicable. Every line begins with the time at which the entry was made. The time is in the format hh.mm.ss.. At the end of the dayfile is a list of the resources used by the job. Possible entries are as follows:

UEAD	Application charge activity in kilounits. This is an overhead charge added to each job.
UEPF	Permanent file activity for the job in kilounits. This activity includes the permanent file operations initiated by the statements SAVE, REPLACE, GET, DEFINE, and so forth.
UEMS	Mass storage activity for the job in kilounits. This activity includes read and write operations, opening and closing of files, and so forth.
UEMT	Magnetic tape activity for the job in kilounits. This activity includes read and write operations, opening and closing of files, and tape positioning.
UECP	Central processor time in seconds.
AESR	System resource units (SRUs) used by the job.
UCLP	The number of lines printed given in kilolines.

The value of a kilounit is determined by the installation. The SRU is an accounting unit that is a composite of central processor time, I/O activity, and memory used.

The user can save an abbreviated version of the dayfile in a separate file by including the DAYFILE control statement in his job's control statement record. This version of the dayfile begins with the job statement and ends with the DAYFILE statement.

The DAYFILE statement can also be entered under the BATCH subsystem in a time-sharing operation from a terminal. This produces a history of time-sharing activity rather than job processing (refer to section 9).

A batch job submitted as a card deck is shown on the left side of figure B-1. The dayfile resulting from processing of the job is shown on the right side.

```
DAYDEMO.
USER(ED25,ED)
CHARGE(5822,NOS5)
NOEXIT.
FTN.
LGO.
LIMITS.
CATLIST.
GET(FILE5)
COPYSBF(FILE5,)
7/8/9
┌──────────────┐
│ FORTRAN program │
└──────────────┘
7/8/9
┌──────────────┐
│ data for the │
│ FORTRAN program │
└──────────────┘
6/7/8/9
```

```
                    jobname.   yy/mm/dd.    installation ID.
08.32.50.DAYDEMO.
08.32.50.USER(ED25,ED)
08.32.50.CHARGE(5822,NOS5)
08.32.50.NOEXIT.
08.32.51.FTN.
08.32.52.       .047 CP SECONDS COMPILATION TIME
08.32.52.LGO.
08.32.52.    STOP
08.32.53.       .008 CP SECONDS EXECUTION TIME
08.32.54.LIMITS.
08.32.54.CATLIST.
08.32.54. CATLIST COMPLETE.
08.32.54.GET(FILE5)
08.32.54.COPYSBF(FILE5,)
08.32.54. END OF INFORMATION ENCOUNTERED.
08.32.55.UEAD,      0.002KUNS.
08.32.55.UEPF,      0.019KUNS.
08.32.55.UEMS,      1.923KUNS.
08.32.55.UECP,      0.618SECS.
08.32.55.AESR,      3.371UNTS.
08.35.03.UCLP, 34,    0.48 KLNS.
```

Figure B-1. Batch Job on Cards and the Resulting Dayfile

The user can obtain a listing of the maximum system resources that the installation has allotted to him by including the following control statement in a job.

LIMITS.

The output from the job will contain a listing of resource limits for the user number. If a value of UNLIMITED is specified for a particular resource, the user has no restriction on usage of that item. If the value is SYSTEM, then this is a default specified by the installation. Numeric values are decimal unless followed by a B to indicate octal.

Example:

A user with user number EFD2511 obtains the following listing of his resources.

```
1          LIMITS.                yy/mm/dd. hh.mm.ss.

 EFD2511        2511     yy/mm/dd. yy/mm/dd.

       AB =,
       AB =,
       AB =,
       AB =,
   MT =              3,
   RP =              2,
   TL =     UNLIMITED,
   CM =          2037B,
   NF =             40,
   DB =             10,
   FC =         SYSTEM,
   CS =          32768,
   FS =         SYSTEM,
   PA = EVEN          ,
   RO =         SYSTEM,
   PX = HALF          ,
   TT = TTY           ,
   TC = STANDARD      ,
   IS = NULL          ,
   MS =          12800,
   DF =            464,
   CC =            464,
   OF =              4,
   CP =           2112,
   LP =          31232,
   EC =             OB,
   SL =     UNLIMITED,
       CN =,
       PN =           ,
   DS =            512,


   AW = 0000000000000000415
```

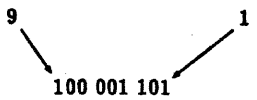
The first line gives the current date and time the listing was produced. The second line gives the user number, user index, date when validation for this user was first established, and the date when validation for this user was last modified.

The fields are defined as follows:

Field	Description
AB	Answerback identifier (1 to 10 alphanumeric characters). Terminals with answerback capability automatically send this identifier to the system when a user logs in. The system uses the answerback to determine if the log-in is from a legal terminal. Each user can have up to four terminal answerback identifiers. In this example, all four are blank implying the installation is not using the answerback capability.
MT	Maximum number of magnetic tapes the user can have assigned to his job at one time. The user in this example can have three.
RP	Maximum number of auxiliary devices the user can have assigned to his job at one time. An auxiliary device is a self-contained permanent file device that is not part of a system configuration of devices (family). This example specifies a limit of two.
TL	Maximum amount of central processor time available for any one of the user's job steps. In this example, there is no limit.
CM	Maximum number of central memory words the user may request for any one job. This is the maximum field length his job can have. The specification is given in multiples of 100B words. The value 2037B implies 203700 octal words of memory, which is 67520 decimal.
NF	Maximum number of files the user can have with one job at one time. The user in this example can have 40 (decimal).
DB	Maximum number of deferred batch jobs the user can have in the system at one time (refer to section 9). In this example, the user can have 10 (decimal).
FC	Maximum number of permanent files the user can have. The example specifies SYSTEM, which means the installation will determine this value.

Field	Description
CS	Maximum number of PRUs available for indirect-access permanent files. A PRU is a physical record unit which on mass storage is 640 6-bit characters. This example specifies 32768 decimal.
FS	Maximum number of PRUs available for any indirect-access permanent file. In this example, this is determined by the installation.
PA	Refers to ODD or EVEN parity in connection with terminal usage. In this example, it is EVEN.
RO	Rubout characters used to delay carriage return at a terminal.
PX	Specifies full- or half-duplex transmission mode for a terminal.
TT	Specifies type of terminal available to the user. TTY is ASCII with standard print. A correspondence code terminal with standard print is COR; a correspondence code terminal with APL print is CORAPL.
TC	Character set to be used by a time-sharing terminal. This is either STANDARD (64-character) or ASCII (128-character).
IS	Initial subsystem for a time-sharing terminal. This is the subsystem that will be in effect when the user does not specify one. Possible values are NULL, BASIC, FTN, EXECUTE, or BATCH.
MS	Maximum number of mass storage PRUs the user may additionally allocate via his job. The user in this example can allocate an additional 12800 PRUs.
DF	Maximum number of messages the user can issue to the dayfile. Refer to section 11 of the NOS Reference Manual, Volume 2.
CC	Maximum number of batch control statements the user can have in his job. The user in this example can have 464 (decimal) statements in his control statement record.
OF	Maximum number of files the user can dispose to output.
CP	Maximum number of cards that can be punched from the user's punch file.
LP	Maximum number of lines a user's job may print. The value 31232 in this example is decimal.

Field	Description
EC	The maximum number of extended core memory words the user may request if the installation is using ECS (extended core storage).
SL	The maximum number of SRUs (system resource unit) the user is allowed for each job. The SRU is an accounting value which the system determines by formula.
CN	Charge number to which the user is assigned.
PN	Project number to which the user is assigned.
DS	Maximum number of PRUs available to the user for any direct access permanent file.
AW	Access word which specifies the user's access options within the system. This is a 60-bit word. When a bit is set, it indicates an associated option is available. At present, 13 options are defined. These are read from right to left. The access word is displayed in octal. In this example, the octal value 415 gives the following bit representation.



This indicates options 1, 3, 4, and 9 are in effect. The following options are of interest to the user of this guide.

Option	Specification
1	User can change his password.
3	User is allowed to create direct access files.
4	User is allowed to create indirect access files.
7	User may request nonallocatable devices (that is, magnetic tape units).
8	User is allowed to access the system without charge or project numbers.
9	User can define, save, and replace files on auxiliary devices.

CDC GRAPHIC	ASCII GRAPHIC SUBSET	DISPLAY CODE	HOLLERITH PUNCH (026)	EXTERNAL BCD CODE	ASCII PUNCH (029)	ASCII CODE
:†	:	00†	8-2	00	8-2	3A
A	A	01	12-1	61	12-1	41
B	B	02	12-2	62	12-2	42
C	C	03	12-3	63	12-3	43
D	D	04	12-4	64	12-4	44
E	E	05	12-5	65	12-5	45
F	F	06	12-6	66	12-6	46
G	G	07	12-7	67	12-7	47
H	H	10	12-8	70	12-8	48
I	I	11	12-9	71	12-9	49
J	J	12	11-1	41	11-1	4A
K	K	13	11-2	42	11-2	4B
L	L	14	11-3	43	11-3	4C
M	M	15	11-4	44	11-4	4D
N	N	16	11-5	45	11-5	4E
O	O	17	11-6	46	11-6	4F
P	P	20	11-7	47	11-7	50
Q	Q	21	11-8	50	11-8	51
R	R	22	11-9	51	11-9	52
S	S	23	0-2	22	0-2	53
T	T	24	0-3	23	0-3	54
U	U	25	0-4	24	0-4	55
V	V	26	0-5	25	0-5	56
W	W	27	0-6	26	0-6	57
X	X	30	0-7	27	0-7	58
Y	Y	31	0-8	30	0-8	59
Z	Z	32	0-9	31	0-9	5A
0	0	33	0	12	0	30
1	1	34	1	01	1	31
2	2	35	2	02	2	32
3	3	36	3	03	3	33
4	4	37	4	04	4	34
5	5	40	5	05	5	35

3AE13A

1 TWELVE OR MORE ZERO BITS AT THE END OF A 60-BIT WORD ARE INTERPRETED AS END-OF-LINE MARK RATHER THAN TWO COLONS.

CDC GRAPHIC	ASCII GRAPHIC SUBSET	DISPLAY CODE	HOLLERITH PUNCH (026)	EXTERNAL BCD CODE	ASCII PUNCH (029)	ASCII CODE
6	6	41	6	06	6	36
7	7	42	7	07	7	37
8	8	43	8	10	8	38
9	9	44	9	11	9	39
+	+	45	12	60	12-8-6	2B
-	-	46	11	40	11	2D
*	*	47	11-8-4	54	11-8-4	2A
/	/	50	0-1	21	0-1	2F
(	(	51	0-8-4	34	12-8-5	28
)	)	52	12-8-4	74	11-8-5	29
$	$	53	11-8-3	53	11-8-3	24
=	=	54	8-3	13	8-6	3D
BLANK	BLANK	55	NO PUNCH	20	NO PUNCH	20
,(COMMA)	,(COMMA)	56	0-8-3	33	0-8-3	2C
.(PERIOD)	.(PERIOD)	57	12-8-3	73	12-8-3	2E
≡	#	60	0-8-6	36	8-3	23
[	[	61	8-7	17	12-8-2	5B
]	]	62	0-8-2	32	11-8-2	5D
%†	%	63	8-6	16	0-8-4	25
≠	"(QUOTE)	64	8-4	14	8-7	22
→	_(UNDERLINE)	65	0-8-5	35	0-8-5	5F
∨	!	66	11-0	52	12-8-7	21
∧	&	67	0-8-7	37	12	26
↑	'(APOSTROPHE)	70	11-8-5	55	8-5	27
↓	?	71	11-8-6	56	0-8-7	3F
<	<	72	12-0	72	12-8-4	3C
>	>	73	11-8-7	57	0-8-6	3E
≤	@	74	8-5	15	8-4	40
≥	\	75	12-8-5	75	0-8-2	5C
¬	~(CIRCUMFLEX)	76	12-8-6	76	11-8-7	5E
;(SEMICOLON)	; (SEMICOLON)	77	12-8-7	77	11-8-6	3B

3AE6A

† IN INSTALLATIONS USING THE CDC 63-GRAPHIC SET, DISPLAY CODE 00 HAS NO ASSOCIATED GRAPHIC OR HOLLERITH CODE; DISPLAY CODE 63 IS THE COLON(8-2 PUNCH). THE SELECTION OF THE 63- OR 64-CHARACTER SET FOR TAPES IS AN INSTALLATION OPTION.

The following operators form expressions in the major higher-level languages and the NOS control language statements.

ARITHMETIC OPERATORS

Character	Operation
+	Addition
-	Subtraction
*	Multiplication
** or ↑	Exponentiation
leading -	Negation
leading +	Ignored

These characters are the same for printer and terminal except that ↑ on a printer is ' (apostrophe) on a terminal.

RELATIONAL OPERATORS

Printer (CDC Graphic)	Terminal (ASCII)	Alphabetic Symbol	Meaning
=	=	.EQ.	Equal to
≠	"	.NE.	Not equal to
<	<	.LT.	Less than
>	>	.GT.	Greater than
≤	@	.LE.	Less than or equal
≥	/	.GE.	Greater than or equal

BOOLEAN OPERATORS

Printer (CDC Graphic)	Terminal (ASCII)	Alphabetic Symbol	Meaning
≡	#	.EQV.	Equivalence
V	!	.OR.	Inclusive OR
Λ	&	.AND.	AND
↓	?	.EOR.	Exclusive OR
¬	^	.NOT.	Complement

EVALUATION OF EXPRESSIONS

Expressions are evaluated according to the following hierarchy.

1. Exponentiation
2. Multiplication, division
3. Addition, subtraction, negation
4. Relations
5. Complement
6. AND
7. Inclusive OR
8. Exclusive OR, equivalence

INDEX

AESR B-1
Alternate access of permanent files 5-4
APPEND control statement 5-2
Arithmetic operators E-1
ATTACH control statement 5-4

Backward positioning statements 4-4
Batch
  Conversational 3-1
  Deferred 3-1
  Local 3-1
  Remote 3-1
Batch input from a terminal 9-1
Batch jobs 3-1
BKSP control statement 4-4
Blocks, tape 8-2
Boolean operators E-1

CALL control statement 6-3
CATLIST control statement 6-3
CHANGE control statement 5-6
Character set, NOS D-1
CHARGE control statement 3-4
Coded copy statements 4-2
Comments, control statement record 3-4
Control language 6-1
Control language statements
  CALL 6-3
  DISPLAY 6-2
  GOTO 6-2
  IF 6-3
  SET 6-1
Control statement format 3-2
Control statement record 3-3
Control statements
  APPEND 5-2
  ATTACH 5-4
  BASIC 3-5,7
  BKSP 4-4
  CATLIST 5-6
  CHANGE 5-6
  CHARGE 3-4
  COBOL 3-5,7
  COBOL5 3-5,7
  COPYBF 4-1
  COPYBR 4-1
  COPYCF 4-2
  COPYCR 4-2
  COPYEI 4-1
  COPYSBF 4-2
  DAYFILE 9-3
  DEFINE 5-3
  EXIT 7-1
  ENQUIRE 9-3
  FTN 3-5,7
  GET 5-2
  job 3-3
  LABEL 8-2
  lfn 3-6
  LIMITS C-1
  NOEXIT 7-1
  ONEXIT 7-1
  PURGE 5-4
  REPLACE 5-2
  RESOURC 8-3
  REWIND 4-5
  ROUTE 9-5
  SAVE 5-1
  SKIPEI 4-4
  SKIPF 4-4
  SKIPFB 4-4
  SKIPR 4-4
  SUBMIT 9-2
  USER 3-3
  VERIFY 4-3
Conversational batch 3-1; 9-4
Copy statements
  Binary 4-1
  Coded 4-2
COPYBF control statement 4-1
COPYBR control statement 4-1
COPYCF control statement 4-2
COPYCR control statement 4-2
COPYEI control statement 4-1
COPYSBF control statement 4-2

Dayfile B-1
DAYFILE control statement 9-3
Deferred batch 3-1; 9-1
DEFINE statement 5-3
Delimiters 2-1
Density, tape 8-1
Direct access permanent file 5-3
Directives, reformatting 9-1
DISPLAY control statement 6-2

End-of-file (EOF) 2-3
End-of-information (EOI) 2-3
End-of-record (EOR) 2-1
ENQUIRE control statement 9-3
EOF 2-3
EOI 2-3
EOR 2-1
Error control 7-1
Error messages, tape 8-4
EXIT control statement 7-1

File
  INPUT 2-3
  Local 2-3
  OUTPUT 2-3
  Permanent 2-3; 5-1
  Tape 8-1

FILE function 6-1
Files, general 2-1
File positioning statements 4-4
Format, control statement 3-2

GET control statement 5-2
GOTO control language statement 6-2

IF control language statement 6-3
Indirect access permanent files 5-1
Input file 2-3

Job control statement 3-3
Job structure, batch 3-1

LABEL control statement 8-2
Labels, tape 8-2
LIMITS control statement C-1
Listing permanent files 5-6
Local batch 3-1
Local files 2-3

NOEXIT control statement 7-1

ONEXIT control statement 7-1
Operators E-1
Output file 2-3

Parity, tape 8-1
Permanent files
    Alternate access 5-4
    Direct access 5-3
    Indirect access 5-1
    Listing 5-6
    Purging 5-4
Positioning statements
    Backward 4-4
    Forward 4-4
Programs 3-5
PURGE control statement 5-4
Purging permanent files 5-4

Recording mode, tape 8-1
Reformatting directives 9-1
Relational operators E-1
Remote batch 3-1
REPLACE control statement 5-2
RESOURC control statement 8-3
Resources, user C-1
REWIND control statement 4-5
ROUTE control statement 9-5

SAVE control statement 5-1
SET control language statement 6-1
SKIPEI control statement 4-4
SKIPF control statement 4-4
SKIPFB control statement 4-4
SKIPR control statement 4-4
Statements
    Control (refer to control statements)
    Control language (refer to control language statements)
SUBMIT control statement 9-2

Tape error messages 8-4
Tape files 8-1
Tape format 8-2
Tape labels 8-2
Tape tracks 8-1
Terminal, batch input 9-1

UCLP B-1
UEAD B-1
UECP B-1
UEMS B-1
UEPF B-1
USER control statement 3-3

VERIFY control statement 4-3
Volume, tape 8-2
Volume serial number (VSN) 8-2
VSN 8-2

COMMENT SHEET

MANUAL TITLE CDC NOS Version 1

Batch User's Guide

PUBLICATION NO. 60436300 REVISION A

FROM: NAME:

BUSINESS ADDRESS:

CUT ALONG LINE

AA3419 REV. 7/75 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD

FOLD