60495100

CONTROL DATA CORPORATION

INTERCOM VERSION 4 INTERACTIVE GUIDE FOR USERS OF COBOL

CDC® OPERATING SYSTEM:
NOS/BE

REVISION RECORD

REVISION	DESCRIPTION
A	Original release.
(1-30-76)	
B	Corrects various technical and typographical errors and reflects PSR summary 473.
(6-20-78)	
Publication No. 60495100	

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 the 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 of 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
Front Cover	-				
Title Page	-	3-8	B	5-1	A
ii	B	3-9	B	5-2	A
iii	B	3-10	B	5-3	A
iv	B	3-11	A	5-4	A
v	B	3-12	A	5-5	A
vi	B	3-13	A	5-6	A
vii	A	3-14	A	5-7	A
viii	A	3-15	A	5-8	A
ix	A	3-16	A	5-9	A
1-1	A	3-17	A	5-10	B
1-2	A	3-18	A	5-11	A
1-3	B	4-1	A	5-12	A
1-4	A	4-2	A	5-13	A
1-5	A	4-3	A	5-14	A
1-6	B	4-4	A	5-15	A
1-7	B	4-5	B	5-16	A
1-8	A	4-6	B	5-17	B
1-9	B	4-7	A	6-1	A
1-10	B	4-8	B	6-2	A
2-1	A	4-9	B	6-3	A
2-2	A	4-10	A	6-4	A
2-3	A	4-11	A	6-5	A
2-4	A	4-12	B	6-6	A
2-5	A	4-13	A	6-7	B
2-6	B	4-14	A	6-8	B
2-7	A	4-15	A	6-9	B
2-8	A	4-16	A	6-10	A
2-9	A	4-17	B	6-11	A
2-10	A	4-18	A	6-12	A
2-11	B	4-19	A	6-13	A
3-1	B	4-20	B	6-14	A
3-2	B	4-21	A	6-15	A
3-3	B	4-22	A	6-16	A
3-4	A	4-23	A	6-17	A
3-5	B	4-24	A	6-18	A
3-6	A	4-25	A	6-19	A
3-7	A	4-26	A	6-20	A

Page	Rev
6-21	A
6-22	A
6-23	A
6-24	A
6-25	B
6-26	A
6-27	A
6-28	A
6-29	A
6-30	A
6-31	A
6-32	A
6-33	A
6-34	A
6-35	B
6-36	A
6-37	A
6-38	A
6-39	A
6-40	A
6-41	A
6-42	A
6-43	A
6-44	A
6-45	A
6-46	B
6-47	A
6-48	B
6-49	A
6-50	A
A-1	A
A-2	A
B-1	A
C-1	B
C-2	B
Index-1	A
Index-2	A
Index-3	A
Cmt Sheet	B
Back Cover	–

Page	Rev

Page	Rev

PREFACE

This manual describes access to CDC® CYBER 170 Series, CDC® CYBER 70 Series, Models 72, 73, and 74, or CDC® 6000 Series computer systems through a remote terminal under control of the INTERCOM Version 4 facilities of NOS/BE 1. Not all INTERCOM capabilities are described; nor are all described commands outlined in full. Throughout the manual, the emphasis is on specific steps a user must take to achieve a final result. The text assumes that INTERCOM is being accessed through a terminal and that batch terminal or central site facilities are not close at hand.

This manual is an introduction to INTERCOM for the COBOL programmer. Its aim is to help those who are writing compiler language programs to use INTERCOM effectively. The EDITOR feature of INTERCOM, which allows a program to be created and updated line by line, dominates the text.

The manual is not written for a programmer experienced in operating system usage who is trying to duplicate batch job execution by a series of commands at a terminal. Nevertheless, those persons may find this manual helpful in reinforcing facts basic to any INTERCOM use.

The first section of this manual contains a concise summary of procedures for accessing the central site and entering and executing a COBOL program.

Section 2 reviews terminal operations.

Section 3 defines the concept of a command and its syntax, as well as the logic behind required user actions and the interaction between EDITOR and other parts of the system. Skip this section if you want a demonstration of EDITOR use before studying EDITOR operation.

Sections 4 and 5 explain how to enter and execute a COBOL program through EDITOR, starting with a call to EDITOR and ending with execution of several types of files. Commands are introduced as they are needed to accomplish a specific task with the minimum parameters possible. Several variations in processing are presented. Full command names are used at all times to reinforce the command or parameter that performs a particular function.

Section 6 presents commands in alphabetical order, discussing capabilities bypassed in sections 4 and 5. Illustrations show situations in which the command is used with other commands to perform specific tasks.

COBOL Version 4 is the language used for examples.

Execution of FORTRAN and ALGOL programs through EDITOR differs in minor details, but operating principles for these languages are the same.

Other manuals containing information that may be useful to INTERCOM users are:

Control Data Publication	Publication Number
INTERCOM Version 4 Reference Manual	60494600
INTERCOM Version 4 Remote Batch User's Guide	60496100
NOS/BE 1 Reference Manual	60493800
COBOL Version 4 Reference Manual	60496800
COBOL Version 5 Reference Manual	60497100
INTERCOM Interactive Procedures Guide	60495200

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

SO YOU WANT TO EXECUTE A COBOL PROGRAM THROUGH INTERCOM? 1

First determine whether your terminal is ready for use by following the procedures below. If it is not ready, turn to section 2 to learn how to connect the terminal with the central site. Otherwise, follow the instructions below and enter your program.

Type the letter Q on the terminal keyboard, then press the RETURN key[†]. If nothing happens, go to the beginning of section 2.

If the following message appears, turn to the LOGIN command heading in section 2 for instructions:

```
    PLEASE LOGIN
```

If the system responds with the word COMMAND, you are connected to INTERCOM. You will want to call EDITOR before entering a program.

ENTER AND EXECUTE PROGRAM

CALL EDITOR

To call EDITOR, type the word EDITOR with no spaces between letters and press the RETURN key. The line entered will look like this:

$$\overbrace{\texttt{COMMAND-}}^{\text{System displays}}\ \underbrace{\underline{\texttt{EDITOR}}}_{\text{you enter}}$$

System response will be two dots at the left of the next line. These dots appear whenever EDITOR is ready for a new command. If two dots appear at the left, EDITOR has been called and you can start program creation with the FORMAT and CREATE commands.

Each remote terminal using EDITOR is assigned one scratch file called the edit file. The edit file is used to create, modify, and examine single text lines, a range of text lines (up to a complete file), or character strings within text lines which may subsequently be submitted for compilation and execution. The edit file is the unstated object of all file-oriented EDITOR commands; that is, the LIST command lists one or more lines of the edit file, the CREATE command places data entered from the keyboard into the edit file, and the RUN

[†]This manual assumes you have a Teletype or a keyboard duplicating a Teletype. For a terminal with a display screen, press the SEND, RETURN, or ETX key, whichever is available. For all RETURN key references, substitute the proper key.

command submits the contents of the edit file for compilation. Normally, all program text is entered into the edit file, reviewed, corrected, and submitted for compilation through EDITOR. When the file has been created, it may be allocated to mass storage, as either a temporary or permanent file, through EDITOR, thus making the edit file available for further use.

CALL FORMAT

The FORMAT feature of EDITOR makes it easier to enter a program by providing a tabbing capability. EDITOR stores a number of predefined formats, and the one that you want is the COBOL format. Call the format by entering:

```
..FORMAT,COBOL
```

System response to EDITOR call

EDITOR acknowledges the request by again displaying the two dots. It does not display the format; that is internal to the system.

If you enter a FORMAT,SHOW command, the system displays the format:

YOUR INPUT

```
                ..FORMAT,SHOW
CH=72 TAB CHAR=;  TAB COL=8 12 16 20 24
```

SYSTEM RESPONSE

As you can see, the format provides you with tab capabilities, just as you would have on a typewriter or keypunch. Unlike these machines, your tab is initiated by entering a character rather than pressing a function key. The tab positions correspond to the most commonly used COBOL columns. (This format may be changed by your installation; therefore, it would be wise to display the format to be certain that it matches the one in this manual.

CALL CREATE

Inform EDITOR that you want to create a program by entering the command CREATE. The system responds by displaying a line number. Your input and system output looks like this:

System response when in EDITOR

```
..CREATE
 100=
```

System response to your CREATE entry

Type the first line in your program after the line number.

You may want to enter and run the program in figure 1-1. It is a complete COBOL program and its execution displays a line at the terminal. Note that the tab character (;) is used to position the entry on the line.

At the end of each COBOL statement, press the RETURN key. Wait for the system to respond with a line number before entering another statement.

A typing error made before RETURN is pressed can be corrected. Hold down CTRL and press the character H once for each character to be erased. Then enter the correct characters.†

A typing error discovered after RETURN is pressed, but before another line is entered, should be corrected by reentering the line correctly. The bad line can be deleted after the entire program is entered.

After all COBOL program statements are entered, enter an equal sign, =, as the first and only character in a line. (Don't forget to press RETURN.) This signals the end of program creation; the system responds once more with two dots.

CHECK INPUT

Verify you have entered the program correctly by calling for a list of the program. Enter this command and press RETURN.

```
. . LIST,ALL     or     . . L,A
```

System response is a list of the program statements input, as shown in figure 1-1. Any line in which the CTRL and H keys were used should appear correct.

CORRECT INPUT

To delete lines, enter DELETE (or D), the line number of the line, and press the RETURN key as follows:

```
. . DELETE,number     or     . . D,number
```

To delete two or more consecutive lines, enter DELETE (or D), the line number of the first line, the line number of the last line, and press the RETURN key. If further changes are to be made, retype any line in error: enter the line number, an equal sign, the correct text, and press the RETURN key as follows:

```
. . linenum=new text
```

The following example illustrates this.

```
. . 180=;;DISPLAY "THIS PROGRAM RUN THROUGH EDITOR" UPON HERE.
190=;;STOP RUN.
```

†On some terminals, the BKSPACE or back arrow key can be used instead of CTRL/H.

```
..CREATE
   100=;ID DIVISION.
   110=;PROGRAM-ID. EXAMPLE.
   120=;ENVIRONMENT DIVISION.
   130=;CONFIGURATION SECTION.
   140=;SPECIAL-NAMES. TERMINAL IS HERE.
   150=;DATA DIVISION.
   160=;PROCEDURE DIVISION.
   170=;FIRST-PARAGRAPH.
   180=;;DISPLAY "THIS PROGRAM RUN THROUGH EDITOR" UPON HERE.
   190=;;STOP RUN.
   200==
..LIST,ALL
   100= ID DIVISION.
   110= PROGRAM-ID. EXAMPLE.
   120= ENVIRONMENT DIVISION.
   130= CONFIGURATION SECTION.
   140= SPECIAL-NAMES. TERMINAL IS HERE.
   150= DATA DIVISION.
   160= PROCEDURE DIVISION.
   170= FIRST-PARAGRAPH.
   180=     DISPLAY ≠THIS PROGRAM RUN THROUGH EDITOR" UPON HERE.
   190=     STOP RUN.
..
```

Figure 1-1. Entering a Sample COBOL Program

To insert any line you may have omitted, use linenum=text, choosing a line number that inserts the line in the proper location.

The two dots do not appear when a correction is made by a command in the format linenum=text. Continue with the next operation after the carriage returns to the left.

Use the LIST,ALL command to examine the program after all changes are made.

EXECUTE PROGRAM

When the program appears as you want it, execute the program by entering this command followed by RETURN key.

```
..RUN,COBOL
```

System response is a status message giving the number of errors, the amount of core memory required for compilation, and the number of seconds the central processor used to compile the program, followed by the line printed as a result of the execution of the program in figure 1-1.

```
COMPILING EXAMPLE
 000 E AND      T/U  DIAGNOSTICS ISSUED
    053277B SCM USED
      .230 CP SECONDS COMPILATION TIME
END COBOL
THIS PROGRAM RUN THROUGH EDITOR
..
```

SAVE PROGRAM

Now that you have confirmed that the program works correctly by executing it, you may want to save it for future use. Any file (or program) in the edit file can be saved by using the SAVE command.

SAVE loads the edit file into a local file whose name is specified as a parameter of the command. (A local file is one which is available only to you and only during the current session.) To keep the program shown in figure 1-1 as a local file named EXAMPLE, enter:

```
. . SAVE,EXAMPLE
```

Notice that the edit file still contains the program even though it has been copied to the local file. This feature allows you to make modifications to a program during debugging while still retaining previous versions.

SUMMARY OF OPERATIONS

1. Establish link to central site if necessary (section 2).
2. Call the editor program: EDITOR
3. Specify format: FORMAT,COBOL
4. Tell the editor that you want to create a program: CREATE
5. Enter program line by line; after all program lines have been entered, exit from CREATE with =.
6. Examine input: LIST,ALL
7. Correct program: DELETE,line and/or linenum=new text
8. Execute program through COBOL: RUN,COBOL

If execution reveals errors, correct the program and execute again, as in steps 7 and 8.

If you want to keep a copy of the program, use SAVE,name.

Recall CREATE to enter another program.

WHAT'S NEXT?

Now that you have successfully entered and executed the COBOL program in figure 1-1, you are ready to execute your own programs through INTERCOM. All other operations are elaborations or variations of these procedures.

INTERCOM operation principles demonstrated include:

Commands are terminated by RETURN key

System prompts next user input

Status and error messages are displayed for user information

EDITOR allows a program to be entered, modified, and executed

A program created with EDITOR can be stored as a local file with SAVE

At this point, you can go ahead with your own programs using commands given above. To learn easier ways of entering a program and additional commands and options available, read section 3 to learn principles of operation or read section 4 to learn from examples.

Section 3 discusses basic INTERCOM and EDITOR operation, as affected by user inputs. Terminology used in explanations of commands is defined. Commands relevant to a concept are introduced so you can associate practical applications with an idea.

Sections 4 and 5 present a series of complete examples of file creation and use through EDITOR. In each example, pertinent command parameters are described. Alternate methods of performing the same task are shown, and suggestions for ease of use or efficient execution are included.

Section 6 lists INTERCOM and EDITOR commands alphabetically, with all optional parameters included. Again, rather than merely a presentation of the options available, emphasis is placed on when to use an option and how it can be combined with others effectively.

If you are not acquainted with your terminal, consult section 2 to learn the various functions of a Teletype.

Read the remainder of section 1 if you are new to INTERCOM. Answers to the questions are meant to encourage your experimentation with INTERCOM commands.

QUESTIONS NEW USERS ASK

Often, the first few times programmers use INTERCOM, they are unsure of system operations. Students, or others new to any type of terminal or computer operations, often are hesitant to use equipment or to deviate from specific instructions.

Answers to the questions which follow are intended to reassure new users about their use of the system.

CAN I HARM INTERCOM OR OTHER PARTS OF THE COMPUTER SYSTEM?

All users at terminals communicate with other parts of the system through INTERCOM. INTERCOM does not allow access to other user's files or to system files that should be accessed only by system analysts. You may destroy files you created or files attached to your terminal, but you will not interfere with other users or the system. The system is designed to protect itself from intentional or accidental misuse. This is not to say that a determined effort to disrupt the system cannot succeed. Most likely, the worst effect of careless or incorrect use of the system will be wasted time and energy, yours and perhaps other users.

CAN I RUIN MY OWN FILES?

Yes, but you can save copies of files so that you will have backup if you do happen to ruin a file. EDITOR can help you avert destruction of all or part of a file by either warning you of the possibility of losing a file or by giving you the opportunity to reconsider an earlier entry. Be careful and save your files frequently until you are confident that you won't accidentally destroy files you have been working on.

WHAT HAPPENS IF I MAKE A MISTAKE?

Don't worry. Few errors are irrecoverable.

INTERCOM will not execute any command that is not in its instruction repertoire. If you enter a series of characters INTERCOM does not recognize, you will receive a FORMAT ERROR or NO SUCH PROGRAM NAME diagnostic message. INTERCOM checks the parameters of each legitimate command to determine whether they are reasonable; if not, a message is returned to you, but execution does not continue. If you make a syntax error you will be informed and, often, prompted for the correct response.

If you decide an executing command was given in error, you can terminate execution by entering %A. If necessary, suspend output by pressing ESC† before entering %A.

You can end a terminal session, if you decide to start all over again, by logging out. Type BYE to exit from EDITOR, then type LOGOUT.

WHAT IF I DON'T KNOW WHAT TO DO NEXT?

Call the TEACH utility by typing the command TEACH and pressing the RETURN key. It will explain commands available for various functions.

If you can't remember the exact command format, try what you think will work. INTERCOM will detect an incorrect format and perhaps prompt you for the correct syntax or format.

†If your terminal does not have an ESC key, use ALT MODE. Holding down the CTRL key and pressing Z has the same effect.

If you are hopelessly confused and the terminal does not seem to respond to your entries, try BYE then LOGOUT. If you can't log out, you can disconnect the terminal by breaking communication. Pushing the CLR button of a Teletype or hanging up the telephone will cause the system to log you out after a couple of minutes; then system resources are not tied up.

You should probably seek experienced help before you LOGIN again so that terminal time can be used productively.

WHY DOES IT TAKE SO LONG FOR THE SYSTEM TO RESPOND?

Are you sure the system isn't waiting for your response? Did you end every command with a RETURN key? Are you using CREATE or ADD with SUP? Usually, but not always, the system acknowledges your entry by displaying two dots, the word COMMAND, or some other status or error diagnostic. First, press the RETURN key to be sure that you are not delaying execution.

Sometimes system response is much slower than at other times. As the system gets busier, response tends to be slower than if EDITOR is processing only a single terminal. Be patient, and read the answer to the next question for suggestions about improving response time.

CAN I SHORTEN RESPONSE TIME?

Yes, by using the system wisely.

Plan your operations. Don't begin with a call for EDITOR if you do not require EDITOR capabilities for creating and updating files. Response time for commands such as permanent file requests may be less when they are entered directly in Command Mode.

Reduce your input errors by typing commands carefully. In the time required for INTERCOM to detect and display an error condition, it may have been able to successfully execute a command correctly entered.

In general, this manual details both the long and short ways to use a command. Even though the long way, in which INTERCOM prompts user response and the user spells out all commands, is best for learning how to use INTERCOM, the experienced user will want to give the maximum amount of information with a single command.

Finally, consider using INTERCOM early in the morning, at lunch time, or late afternoon. Experience at many installations shows that INTERCOM usage is heaviest around 10 a.m. and 2-4 p.m.

DO I HAVE TO WAIT FOR INTERCOM TO RESPOND BEFORE ENTERING ANOTHER COMMAND?

Usually, no.

INTERCOM executes each command as it is received. For some types of commands, such as a sequence calling for REWIND and COPY, or EDITOR and CREATE, more than one command can be entered; each is executed in turn. An error diagnosed during execution of the first command, however, often prevents successful execution of the second command.

If the command or program requires additional input, don't enter what is not expected. In commands that change a line of text, for example, selection of the VETO option requires YES or NO response to the change displayed. Another command entered before YES or NO will be accepted as if it were the expected input. Often a diagnostic or error message results instead of successful command completion.

WHAT IS A REASONABLE TIME TO WAIT BEFORE SUSPECTING SYSTEM PROBLEMS?

It depends on what you are doing and how busy the system is.

One command often results in many system tasks. The RUN command with a COBOL program, for example, causes a file to be copied, and the copied file to be compiled before a response is returned to the user. Obviously a command such as RUN requires more time than a command to rewind a file, since it involves calls to many parts of the system.

When many people are using the system, response time may be slower. EDITOR itself is affected by the demand for system resources made by other non-INTERCOM jobs. The central site operator can assign priorities to different classes of jobs. Usually, EDITOR jobs run with the highest priority for access to the central processor, but at times, the central site operator may decrease EDITOR priority to accommodate other system users.

HOW DO I KNOW INTERCOM IS ACTIVE OR THAT THE COMMUNICATION LINKS ARE STILL INTACT?

An active system acknowledges a user entry by changing the line or character position marker. If you press the RETURN key at a Teletype, the system moves the paper up one line. If you press the RETURN or SEND key at a CRT terminal, the system responds with line feed; and the line marker at the left will advance by one line.

At terminals connected by acoustic couplers, check that the carrier light is still on. Communication has been lost if the light is off.

If the response time seems inordinately long compared to previous entries, be sure that the system is not waiting for your response. Until a RETURN key is entered, the system assumes your entry is not complete. Sometimes users are distracted and forget to press the RETURN key.

Look at the typing element position. If it is not at the left, check the characters in the line. There may be a message from the system asking for your response; the system does not always position to the left for your entry. If the last line entered is an entry you have created, press RETURN. It may be the first time the system is aware of your entry; if it is the second RETURN, it should not interfere with execution of the current entry.

WHAT HAPPENS IF COMMUNICATION WITH THE CENTRAL SITE COMPUTER IS DROPPED THROUGH NO FAULT OF MINE?

That depends on why communication is down.

All files associated with the terminal are recovered whenever possible. Re-establish communications and LOGIN again with the same name and password to establish your identity. If system response is two dots at the left, the edit file you were using at time of disconnect is intact. Enter the FILES command to determine whether other files are intact. Except under extraordinary circumstances, permanent files are available, although they may have to be attached again.

ASSUMING THE MANUAL TELLS ME WHAT TO DO FOR EACH COMMAND, WHAT SHOULDN'T I DO?

Until you have completed the terminal session with a LOGOUT command or become hopelessly confused, do not break the communication with the central site. Do not push the CLR button or pull the plug on a Teletype, or hang up the telephone or put the telephone on HOLD if you are connected to the system through an acoustic coupler. Breaking communications could cause the loss of files you may have just entered.

Do not expect proper results if you indiscriminately change equipment settings or deliberately violate principles of operation.

COMMUNICATION LINKS AND HARDWARE OPERATION 2

ESTABLISHING INTERCOM ACCESS

To use INTERCOM, the first user at a terminal session normally:

Establishes hardware communication between the terminal and the central site.

Establishes software communication with INTERCOM by entering a LOGIN command.

Hardware communicates with the central site through telephone lines. Most often, these lines can be dialed. Some terminals are hardwired to the central site. For such terminals, LOGIN is the only user action needed to establish communication.

Sometimes, software communication is established at the start of the day or at the beginning of a class period and not disconnected until a scheduled time. The need for a LOGIN command varies among installations.

If someone else has immediately preceded you at a terminal and both hardware and software links to the central site exist, you can enter commands at once.

At a Teletype, to determine whether the terminal is ready to accept commands, check status lights and try a command:

- Look at the light marked ORIG in the lower right area. If it is not lit, hardware communication is not established; and the user must follow procedures for dialing into the central site computer.

- If the ORIG light is on, enter the word FILES and press the RETURN key. If this message is displayed:

  ```
  PLEASE LOGIN
  ```

  software communication is not established and the user must enter the LOGIN command as outlined below.

- If the ORIG light and the BRK-RLS light are on, press the BRK-RLS button before entering the command.

DIALING INTO THE CENTRAL SITE

At a Teletype terminal you can dial into the computer at the central site just as if you were dialing a friend's home. The dial on a Teletype is the same as that on a home phone. Push the ORIG button so that the light comes on, then dial the number you have been instructed to use.

Dialing the number establishes a connection with equipment at the central site. When the line is free for use, you will hear a high pitched tone. If the line is not available, you will hear a busy signal similar to that for any other busy telephone line. The high pitched tone indicates only that the line is available for use between the terminal and the central site equipment. It does not necessarily mean that the computer software programs are in operation.

When telephone connection is established and INTERCOM is operating, the following message is displayed:

```
CONTROL DATA INTERCOM 4.5
DATE  10/08/75
TIME  10.46.26.

PLEASE LOGIN
```

The specific message at your terminal may be different. This one identifies the system in use and gives the current date and time at which hardware communication was established, based on a 24-hour clock.

If you hear the high pitched tone (and have pushed any buttons required for operation of your particular terminal), but the LOGIN message does not appear, press the RETURN key. Response from an active central site computer will be the LOGIN message. If the RETURN key does not activate the message, assume the central site is not currently running INTERCOM.

ACOUSTIC COUPLER CONNECTION

Some terminals must be connected through an acoustic coupler, which includes a cradle for a telephone handset. Examine the cradle to determine which end should receive the mouthpiece and attached cord. Words similar to CORD HERE may appear; if they do not, look for a groove or channel to guide the cord or a diagram showing a handset and cord. The direction in which the handset is placed is significant.

Turn the coupler ON if necessary.

Set any FULL/HALF DUPLEX switch to HALF.

Dial the central site. When the high pitched tone is heard, set the handset firmly in the cradle.

If the terminal does not respond with a request for a LOGIN, press the RETURN key. Placing the handset in the cradle may generate spurious line signals that prevent the message from being sent. Also, check status lights for communication between the coupler and the terminal. Lights on the coupler show communication between the central site and the coupler only; terminal lights show terminal/coupler communication. Perhaps the plug linking the coupler and the terminal is not secure. Make sure the terminal is turned on.

LOGIN COMMAND

Software communication with INTERCOM is established by the LOGIN command. It must be the first command entered after hardware communications are established. At the conclusion of a successful LOGIN procedure, INTERCOM will have:

Verified your right to access the system.

Assigned a 2-character user identifier.

Made access possible to other INTERCOM commands.

The sequence of entries required for LOGIN may vary at your installation. Generally, however, LOGIN asks the user to identify himself and to give a password that indicates he is authorized to use INTERCOM. Sometimes any name, such as your last name, may be acceptable with a given password; other times a particular name is necessary. Often a name is a department or class identification, and the password is an accounting number. Consult with your instructor or system analyst for specific instructions.

The name and password entered during LOGIN must conform to a range of acceptable values stored at the central site. Both of these items may consist of 1-10 letters or digits. If you do not enter values that match allowed values, INTERCOM cannot be used.

Part of the function of the password is to restrict INTERCOM access to authorized users only. Consequently, when INTERCOM asks for a password to be typed, it will prepare an input area on the Teletype paper by overstriking several characters and, then, setting the carriage to the beginning of that line.

```
PLEASE LOGIN
LOGIN
ENTER USER NAME- SVLOPS
$$$$$$$$$$ ENTER PASSWORD-
```

Your typing will be printed on the blacked out area, so it will be illegible, preventing unauthorized persons from learning your password by reading discarded terminal paper.

To begin the LOGIN procedure, type LOGIN and press the RETURN key. Then wait for the system to respond with instructions for additional entries. When prompted, type the item requested, ending each item with a RETURN key.

After all items are entered, successful completion of LOGIN will be marked by the appearance of a message similar to:

```
08/13/75     LOGGED IN AT 14.06.29.
             WITH USER-ID CO
             EQUIP/PORT 25/060

COMMAND-
```

The USER-ID is the means by which INTERCOM identifies the user and files associated with the user. Look for this identification when lists of jobs in the system are being scanned. Also, use this identifier if you need to consult with the central site operator or system analyst. The EQUIP/PORT information shows the hardware connnection to the central site and is not significant for beginners to remember.

At some installations, additional messages may appear before the word COMMAND. Such messages may be system bulletins prepared by installation analysts to inform users as to hours of operation or phone numbers of analysts, as well as instructions to follow for obtaining more help or system information.

You can hasten LOGIN completion by entering some or all of the items required at the same time the word LOGIN is entered. If this option is selected, use commas to separate items.

Any of the following could be entered for LOGIN:

1. User enters the characters LOGIN and waits to be prompted before entering other items

   ```
   LOGIN
   ```

   System asks for a user name, then asks for a password.

2. User enters a user name before pressing the RETURN key.

   ```
   LOGIN,MYNAME
   ```

   System prompts a password entry.

3. User enters both user name and password.

   ```
   LOGIN,MYNAME,SECRET
   ```

   System responds with informative bulletins or messages.

4. User suppresses long system messages by adding the SUP parameter to the LOGIN entry.

   ```
   LOGIN,MYNAME,SECRET,SUP
   ```

The SUP parameter inhibits display of the LOGIN acknowledgment and any other optional system information. Information an installation deems too important for the user to miss appears despite use of SUP.

Once the word COMMAND appears, INTERCOM is ready to receive additional user commands.

Summary of LOGIN:

```
PLEASE LOGIN

LOGIN

ENTER USER NAME -

MYNAME

ENTER PASSWORD -

SECRET

(DATE, TIME, USER-ID)
```

```
(INSTALLATION INFORMATION MAY APPEAR HERE)

COMMAND -
```

Summary of abbreviated entry:

```
PLEASE LOGIN

LOGIN,MYNAME,SECRET,SUP

COMMAND -
```

DIAGNOSTICS FOR INCORRECT USER ACTIONS DURING LOGIN

Messages returned to the terminal indicate difficulties in executing LOGIN because of current system status or user error.

PLEASE LOGIN

LOGIN must be the first command entered after hardware communication is established. This message appears if some other command is entered, or perhaps LOGIN has been misspelled.

LOGIN NOT PERMITTED AT THIS TIME

Push the CLR button, hang up the handset, or otherwise abandon attempts to use INTERCOM at this time. It is not possible for a terminal to begin INTERCOM processing. This condition is initiated by the central site operator and does not necessarily indicate either a system malfunction or a user error. Often the appearance of this message precedes a scheduled interruption of system operation. Currently logged-in users are allowed to complete operations, but no new users can LOGIN.

INVALID USER NAME OR PASSWORD

Your entry of a name or password must match entries on a list at the central site. Check that you have spelled required parameters correctly. Re-enter the correct letters or numbers for the item prompted.

USER NAME/PASSWORD IN USE AT ANOTHER TERMINAL

Be sure you are entering the correct characters. Use another name or password if you have been assigned alternates. Otherwise, consult with the person who originally assigned your LOGIN parameters.

PREVIOUS USER AUTO LOGGED OUT

The last user did not log out. The system has logged-out the last user automatically. No action is required.

YOU HAVE HAD THREE TRIES — GET HELP

The wrong names or passwords have been entered three times. INTERCOM assumes you have the wrong LOGIN information since a simple typing error should not be repeated. Obtain the correct words for INTERCOM access, then begin again with the LOGIN command.

LOGOUT COMMAND

LOGOUT is the last command entered when all work is complete. It terminates communications with INTERCOM and the central site. No further commands can be entered until the LOGIN command is used to re-establish communication.

LOGOUT destroys all local files associated with the terminal except those you have retained as permanent files prior to LOGOUT. (The STORE command will make a local file permanent.) Any files identified as REMOTE INPUT FILES, REMOTE EXECUTING FILES, and REMOTE OUTPUT FILES remain in the system until the user takes specific action to dispose of them. See the discussion of FILES command.

LOGOUT cannot be called when EDITOR is being used. Type BYE to exit from EDITOR before attempting to log out.

```
BYE
```

The command is simply the word LOGOUT followed by a RETURN key.

```
LOGOUT
```

When this command is executed, information summarizing session time is returned.

```
COMMAND- LOGOUT

CPA       2.245 SEC.        2.245 ADJ.
CPB        .000 SEC.         .000 ADJ.
SYS TIME                   23.952
CONNECT TIME     0 HRS.   20 MIN.
 10/08/75  LOGGED OUT AT 11.06.02.
```

CPx indicates how many seconds each central processor was used for all operations during this terminal session.

SYS TIME is computed from a formula that reflects all system resources used.

CONNECT TIME shows the elapsed time, in number of hours and minutes, between the LOGIN command and LOGOUT command execution. These times are used by the system to determine accounting charges for the people using INTERCOM. You need not be concerned with them, unless perhaps you have been asked to keep your own or a class log of terminal work.

The time used by the central processor is small in comparison to that of the peripheral processors. Compilation and execution requires use of the central processor, but most INTERCOM functions, including displaying information at the terminal, occur in peripheral processors.

TELETYPE OPERATION

The various Teletype models have been designed for many applications and often include features and controls irrelevant to INTERCOM operation. The model 33 Teletype is smaller and has fewer features than models 35 and 38. None of the differences in appearance between models affect INTERCOM use. Figures 2-1 thru 2-5 show Teletype models 33, 35, 38 and the buttons used for establishing communication.

ORIG	Activates Teletype
CLR or CLEAR ALARM	Deactivates Teletype
K	For model 35 only, activates the keyboard

The buttons may not be located in the same place on all units, but their functions are the same.

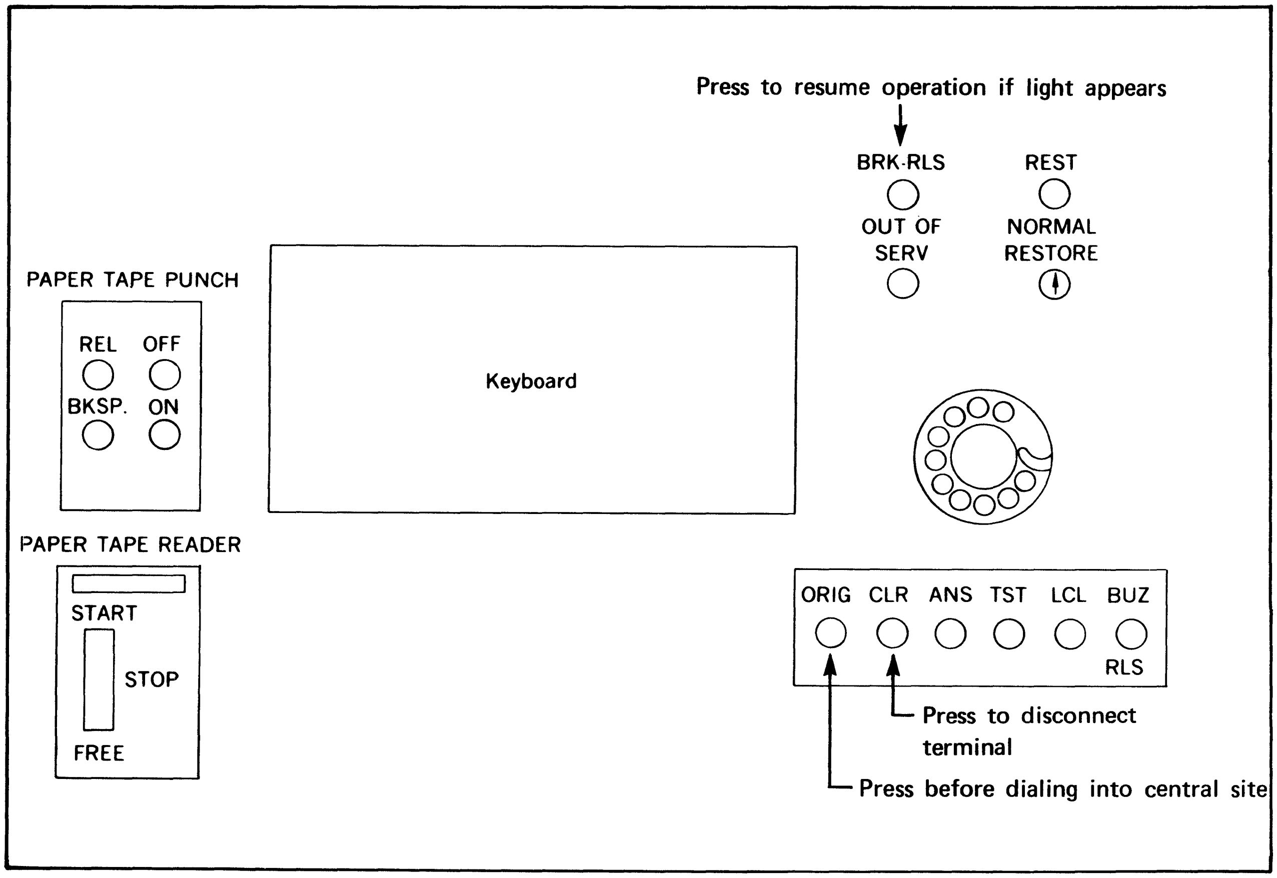

Figure 2-1. Model 33 Teletype

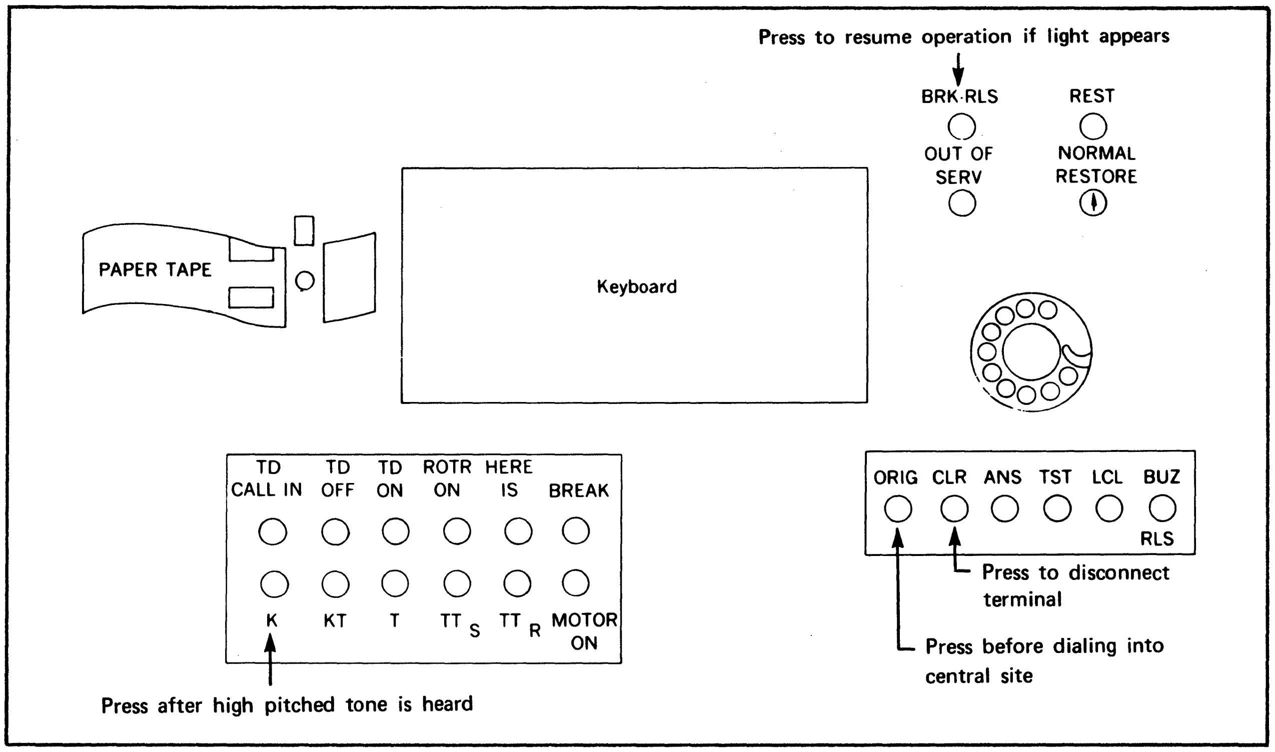
Press to resume operation if light appears
BRK-RLS
REST
OUT OF SERV
NORMAL RESTORE
PAPER TAPE
Keyboard
TD CALL IN
TD OFF
TD ON
ROTR ON
HERE IS
BREAK
K
KT
T
TT S
TT R
MOTOR ON
Press after high pitched tone is heard
ORIG
CLR
ANS
TST
LCL
BUZ RLS
Press to disconnect terminal
Press before dialing into central site

Figure 2-2. Model 35 Teletype

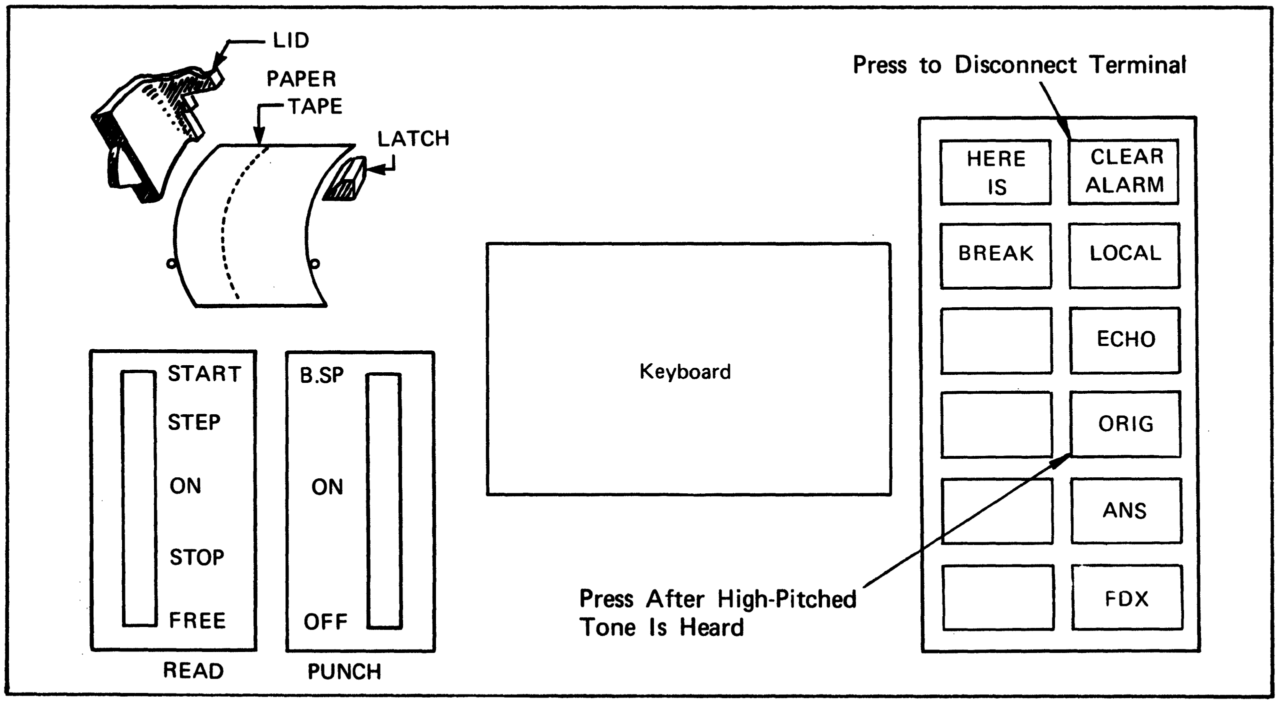

Figure 2-3. Model 38 Teletype

The keyboards are similar to that of a typewriter (figures 2-4 and 2-5), and have a variety of extra keys. The characters differ in placement, but the function is not affected by location.

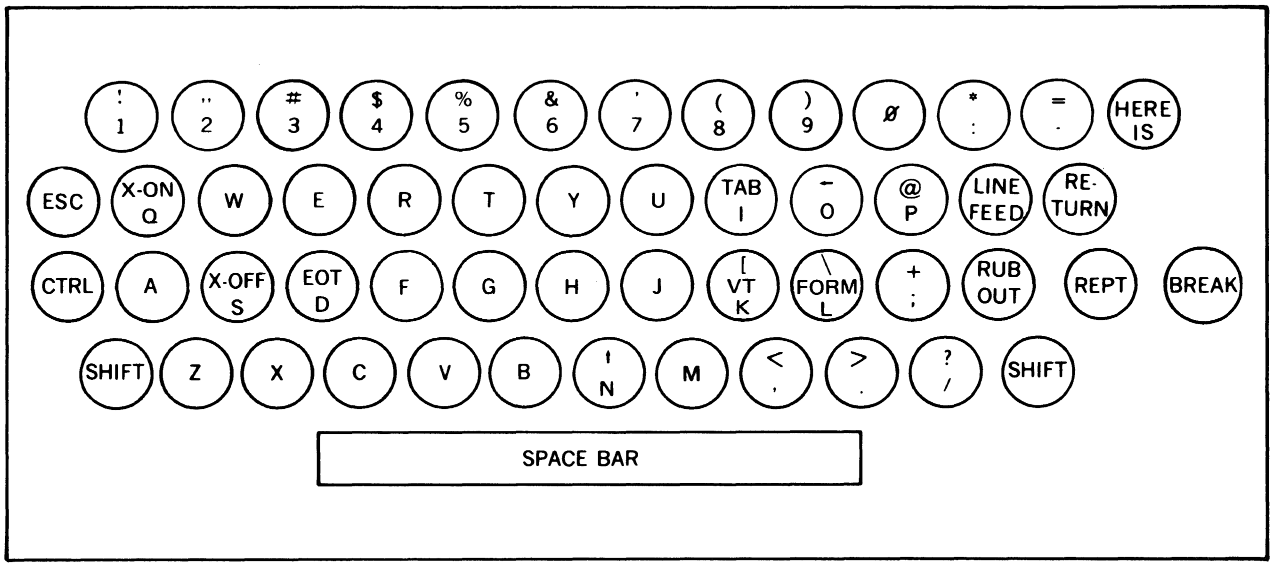

Figure 2-4. Typical Teletype Keyboard, Models 33 and 35

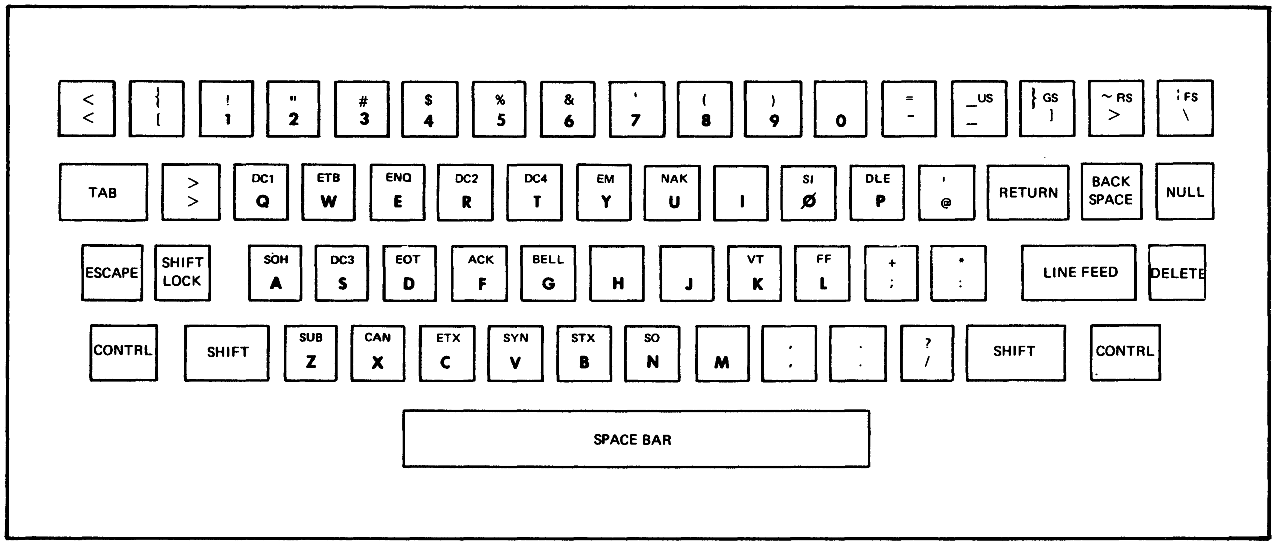

Figure 2-5. Typical Teletype Keyboard, Model 38

The following characters are used for INTERCOM commands:

letters A-Z

digits 0-9

characters = / , ( ) $; .

Any other character defined by the character set in use at the central site (appendix A) may be entered as part of a line in a user file.

The SPACE bar functions as for a typewriter. Display of a line entered shows a blank character for operation of the space bar.

The SHIFT key is used to access some of the upper characters of double keys, as it is on a typewriter.

Several keys significant for INTERCOM use have no counterpart among the keys of a standard typewriter; some of these are the CTRL, LINE FEED, and RETURN.

The RETURN signals the end of an input line. The system advances the paper and moves the printer carriage to the far left. On a typewriter, a carriage return is necessary to confine characters to the physical page. On a Teletype, however, the function of a carriage return is performed by a LINE FEED, not a RETURN.

Although the following two sequences appear the same to a person sitting at a Teletype, their interpretation by INTERCOM is significantly different.

1. Enter FILES and press LINE FEED. Printer carriage returns to far left of next line.

2. Enter FILES and press RETURN. Printer carriage returns to the far left of next line.

In the first instance, INTERCOM is waiting for more user input. LINE FEED is acceptable only as a compensation for physical paper size.

In the second instance, however, INTERCOM executes the FILES command and displays the results of execution. RETURN signals the end of a user command; LINE FEED does not affect a command.

A model 33 or 35 Teletype line has a maximum of 72 characters. A model 38 Teletype uses a maximum of 132 characters. INTERCOM, however, can accept more than 72 characters input as a single line. The significant function for INTERCOM operation is not the line on which an input command appears physically, but whether the RETURN key has been pressed to signal the end of a command.

Each time the user presses a key of a Teletype, the associated character or function is transmitted immediately to the central site. INTERCOM interprets each character as it is received. Possible interpretations are:

The character is a member of the installation character set. It is saved in a current line buffer for future use after a RETURN is received.

The character is TAB, FORM, or some other character not recognized as a valid function. The character is discarded.

The character is a function LINE FEED, RETURN, CTRL, any of which causes INTERCOM response:

LINE FEED	Issues a carriage return.
RETURN	Issues a line feed. Executes command accumulated in the buffer as each character was entered. If a file is being created through EDITOR, the RETURN key signals the end of a line. INTERCOM issues a line feed in response to the carriage return.

CTRL	Takes action on current line buffer according to character H or X pressed while CTRL is pressed.	
	H	Logical character backspace. Each time H is pressed the last character in the buffer is erased.
	X	Logical line backspace. All characters entered by the user since the last RETURN are erased. The carriage is not repositioned.
CTRL and Z[†] (ESC or ALT MODE)	Interrupts command execution. Pressing CTRL and Z together suspends output to the terminal. The next character entered must by %S or %A followed by RETURN or RETURN pressed alone. The interrupt (CTRL/Z, ESC and ALT MODE), must be used prior to entering a %A or %S if output is to be suspended; otherwise the abort or suspend will not be recognized.	
	%S	Stops output. Any data awaiting output to the terminal is discarded. If more output is generated, it will be displayed. %S may suppress the EDITOR . . response or the word COMMAND. Any executing command will be completed.
	%A	User abort. Stops further command execution. Any output destined for display will be discarded. The message USER ABORT is displayed.
	%EOR	End of record. An end of record signal is sent to the executing program.
	%EOF	End of file. An end of file signal is sent to the executing program.
	RETURN	Resumes execution as if CTRL and Z had not been entered. The interrupted logical line is repeated.

Any input from the keyboard interrupts output. If output stops unexpectedly, first check the BRK-REL button. If it is lit, press it to resume operation. If this button is not lit, enter a RETURN. The RETURN will restart the output if it has been stopped by line noise or any other interruption except %A or %S. Line noise is a spurious transmission signal that originates from sources other than a keyboard character entry. It may originate, for example, if the telephone handset connection is bumped or electrical disturbances occur. Also, line noise can occur if the handset cradle of an acoustic coupler does not muffle room noise sufficiently.

[†]When available, the ALT MODE or ESC key should be used in place of CTRL and Z.

CONCEPTS AND PRINCIPLES OF OPERATION 3

INTERCOM is a CDC product that allows a user at a terminal to access the processing facilities of a computer located at a central site some distance away. The central site computer used by INTERCOM may be one of several models of the CDC CYBER 170, CYBER 70, or 6000 Series computer systems. These large computers are capable of handling many different tasks at the same time.

Normally, at the central site, a user submits a program for execution as a deck of punched cards. The first card in the deck – the job statement – identifies the deck and often the person submitting the deck. A series of instructions for the system follows; they are called control statements and typically would call for compilation then execution of a program. The program itself would follow in the deck, along with any data the program would use during execution. The last card in the deck has a special punch combination signaling the end of this job.

This particular job deck, along with many others, is placed into a card reader. From this point, system programs control the progresss of the job from the card reader to mass storage to execution and output of program results. At minimum, each job deck results in a listing on a line printer that is returned to the user to show steps taken while the job was in the computer system.

INTERCOM enables a user at a remote site to perform the tasks on a computer without having that computer, or its peripheral card readers and line printers, physically available.

Many users at many different locations can access INTERCOM at the same time. Simultaneously, other user jobs submitted at the central site are being processed. The user at any given site, however, is not aware of all the jobs in process at once. Rather, he knows only that the system responds to each of his instructions. The user enters an instruction and waits for a response to show that the desired action is completed. During execution, INTERCOM ensures that all input and output from a terminal is routed successfully for execution.

INTERCOM can control operations from several different types of equipment. Terminals commonly used are:

Teletypes

Several different types of portable terminals that look like modified typewriters and operate similarly to Teletypes

CRT terminals (cathode ray tubes similar to television screens)

Batch processing terminals with a CRT and attached card reader, line printer, and/or card punch

The terminals with card readers and line printers attached operate similarly to the card readers and line printers at the central site. They are called batch terminals because they allow a batch of user job decks to be entered into the system and executed just as if they were at the central site. Such terminals are not described here.

The description of terminals that do not have peripheral equipment attached includes how the user communicates with INTERCOM when a card reader and line printer are not available. Further, instructions are given for using file construction and manipulation features that are available only through INTERCOM.

WHAT IS A COMMAND?

A command is a user entry that calls for INTERCOM action. LOGIN, which establishes communication with the central site, is a command, as are FILES and EDITOR.

All commands have similar characteristics:

They have a keyword that identifies the command.

They may have optional parameters separated by commas.

They are not complete until a RETURN key is pressed.

Syntax of the individual commands depends on several variables not readily apparent nor significant to beginning users.[†] If you use commas between parameters, do not include any blanks, and do not terminate the command with a period, you always will have acceptable command formats.

When the EDITOR facilities of INTERCOM are used, command format is more flexible, as discussed in the EDITOR topic below. Since most INTERCOM commands not directly pertinent to EDITOR features can be entered in EDITOR mode, it is easy to be confused about format. Until you are familiar with what is or is not an EDITOR-only command, use a comma between parameters. EDITOR commands must not be terminated by a period.

The last character of an INTERCOM command may, but need not be, a period. INTERCOM will supply one, if necessary, before passing a command to other parts of the operating system for execution.

A command can be entered any time this word is displayed at the far left:

```
COMMAND-
        ↑
        typing element is
        located at this point
```

VALID COMMAND NAMES

INTERCOM has an internal list of valid command names. They can be the names of commands to be executed by INTERCOM itself, or they can be operating system control statements.

Table 3-1 lists some of the commands recognized by INTERCOM. Other commands relevant only to operation of different equipment or to tasks not commonly performed by beginning INTERCOM users have been omitted. Consult the INTERCOM reference manual for additional valid commands.

To avoid confusion in learning how to use INTERCOM, do not create a file with a name duplicating a command name.

[†]Commands corresponding to operating system control statements must conform to the operating system control statement syntax. INTERCOM EDITOR commands need not do so.

TABLE 3-1. VALID INTERCOM COMMANDS, PARTIAL LISTING

Command	Description
BATCH	Changes file category or sends file to another site
DISCARD	Eliminates file or purges permanent file
EDITOR	Calls facilities that create and edit files
FETCH	Accesses permanent file
FILES	Lists files associated with user
LOGIN	Establishes INTERCOM access to begin terminal session
LOGOUT	Terminates terminal session
STORE	Makes file permanent
TEACH	Calls utility that explains INTERCOM use

An additional set of commands, available once EDITOR has been called, is listed in table 3-2, under EDITOR operation, later in this section.

INVALID ENTRY RESPONSES

A user command entry is checked by INTERCOM in several ways:

The first check determines that the command consists of 1 to 7 letters and digits (the first character must be a letter), or 1 to 7 letters and digits followed by a list of parameters.

Then the first word of the entry is checked against a list of valid INTERCOM commands. If it requires INTERCOM action, any parameters are checked before execution.

An operating system command is executed by other parts of the operating system.

If the entry is not a call for INTERCOM or operating system execution, it is assumed to be a call for loading and executing a user file having the entry name.

An entry with invalid characters or too many characters produces the following diagnostic:

FORMAT ERROR

Omission of a required comma separator also may produce these same messages. An entry that produces either diagnostic must be re-entered.

INTERCOM can detect many types of user errors; but a misspelled command name cannot be diagnosed. A misspelled command, however, may produce an error diagnostic that seems unrelated to your entry. If a diagnostic is not meaningful, check that you have entered the command name correctly.

For instance, the following diagnostic might appear when you spell an INTERCOM command incorrectly and enter a sequence that is a valid command from other types of INTERCOM terminals.

COMMAND/TERMINAL MISMATCH

If an entry does not match a command name or an existing user file name, the operating system routines attempting to load and execute the file issue an error diagnostic. Typing FILWA instead of the FILES command, for instance, results in:

```
COMMAND- FILWA


 NO SUCH PROGRAM CALL NAME  -  FILWA
```

$\uparrow$ system inserts name of non-existent file

If the user file exists, but does not contain an assembled program that can be used by the loader, a diagnostic similar to the following appears:

```
COMMAND-SOURLD

55053005032524055555       EXECUTF
23172522030555031704      SOURCE COD
05552711141455161724      E WILL NOT
55053005032524055555       EXECUTE
23172522030555555555      SOURCE

NO PROGRAMS READ YET
LAST FILE ACCESSED- SOURLD
           FATAL LOADER ERROR-
EMPTY LOAD
```

When any of these diagnostics appear, you must re-enter the correct command format. Revert to using a comma as the only separator between words in a command, and you can eliminate the diagnostic for many entries having no other apparent error.

Invalid or missing parameters in an INTERCOM command produce a variety of diagnostics, as discussed in section 6 under each individual command. Often, the message displayed requests a missing parameter; and once that parameter is entered, execution continues.

Another diagnostic referencing internal operating system data can occur for some commands when a comma separator is the last character before the RETURN key is pressed.

```
COMMAND- REWIND,COMB,
 ILLEGAL I/O REQUEST
  FILE NAME
  FET ADDRESS   000112
  ILLEGAL FILE NAME
```

System inserts filename; since the error was blank file name, no name appears.

Ignore this line; it pertains to internal system execution, not a user entry.

The same message can appear if an invalid file name is used as a parameter of the RETURN or REWIND command. The message indicates an error in executing the parameter list, not in the command name. Successful execution occurs for files named before, but not for those named after, the incorrect name.

The following message indicates that you must reenter a command.

REPEAT LINE

Occasionally, when INTERCOM use is heavy, the command input buffer at the central site can become saturated. The message tells you the last command was not accepted.

ABNORMAL COMMAND TERMINATION

Under most circumstances, you anticipate complete execution of a command entered. At times, however, you may want to stop execution before completion. For instance, you may realize after you have requested execution that you have made a logic error in a program; and you may want to terminate the command so as not to waste system resources.

The output of any command execution can be suspended by pressing the ESC† key. The next key pressed, then, determines whether execution resumes. (Remember, do not press the ESC key unless you want to interrupt output.)

Press the % and A keys to abort the command.

Press the % and S keys to discard all output currently waiting transmission to the terminal.

Press the RETURN key to continue the interrupted output.

The RETURN key must be pressed to enter an abort (%A) or suspend (%S) request as with any other INTERCOM command.

System response to receipt of the abort request is the termination of the executing command, whether it is a user program execution or an INTERCOM command such as FETCH. The terminal displays:

USER ABORT

EXAMPLES OF ENTERING A COMMAND

Try several sequences illustrated as follows to become familiar with command entries.

The TEACH command is illustrated. Its format is simply the word TEACH. Execution produces a list of items to be selected for further instruction in INTERCOM use. TEACH is aborted in the same way as any other command, by pressing ESC, if output must be interrupted followed by %A.

†If your teletypewriter does not have an ESC key, use ALT MODE. If that key is not available, hold down the CTRL key and press Z.

1. Normal command entry:

```
          COMMAND- TEACH
                        ↑
                        press RETURN
```

Execution response is:

```
     TEACH
  IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

HOW TO USE INTERCOM                   TYPE   1
HOW TO USE THE TERMINAL               TYPE   2
AN INTERACTIVE COMMAND                TYPE   3
AN EDITOR COMMAND                     TYPE   4
A REMOTE BATCH COMMAND                TYPE   5
%A
↑
ESC, % and A pressed
```

Response to abort request:

```
USER ABORT
COMMAND-
```

2. TEACH is spelled incorrectly:

```
COMMAND- TEECH

NO SUCH PROGRAM CALL NAME  -  TEECH
```

This message appears any time the user entry is not a valid command and a file with the entry name does not exit.

3. One command correctly entered on two physical lines:

```
COMMAND- TEA
CH                   ↑
   ↑                 press LINE FEED
   press RETURN

     TEACH
  IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

HOW TO USE INTERCOM                   TYPE   1
HOW TO USE THE TERMINAL               TYPE   2
AN INTERACTIVE COMMAND                @YP%A

USER ABORT
```

A LINE FEED during command input affects the carriage.
RETURN is the proper end-of-command signal.

4. Same as example 3, but space bar is pressed incorrectly before LINE FEED:

```
COMMAND- TEA
CH            ↑                  ↑
  ↑        space in error     press LINE FEED
  press RETURN

NO SUCH PROGRAM CALL NAME   -   TEA
```

5. Logical equivalent of incorrect example 4:

```
COMMAND- TE   ACH

NO SUCH PROGRAM CALL NAME   -   TE
```

Spaces are not allowed in a command keyword.

6. Spaces before a command cause a diagnostic message to be issued but are ignored after a command.

```
COMMAND-  TEACH

FORMAT ERROR
```

7. RETURN incorrectly used in attempt to duplicate example 3:

```
COMMAND- TEA
CH                 ↑
  ↑             press RETURN in error
  press RETURN

 NO SUCH PROGRAM CALL NAME   -   TEA
COMMAND-
 NO SUCH PROGRAM CALL NAME   -   CH
```

RETURN signals the end of a command.

8. Backspace used correctly on single character:

```
COMMAND- TEEACH
           ↑   ↑
               press RETURN
           press CTRL and H, then A to replace E

     TEACH
  IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

HOW TO USE INTERCOM                  TYPE   1
HOW TO USE THE TERMINAL
```

9. Backspace used correctly for several characters:

```
COMMAND- TEECHACH
              ↑    ↑
              |    press RETURN
              |
              press CTRL and H
              3 times; enter correct
              characters

     TEACH
   IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

HOW TO USE INTERCOM               TYPE    1
HOW TO USE THE TERMINAL       @ %A
```

USER FILES

A file is a collection of information referenced by its file name. A file can contain a program to be run, data to be used by the program, or the output from a program. Also, a file may exist as a name only, having no information at present, but available for writing.

When you reference a file name in a command, you indicate that all information in the file is to be handled as a unit.

A user has control over any file associated with his terminal. These files may have been:

Created through the EDITOR facility of INTERCOM and saved as files with specific names

Stored as permanent files and retrieved by FETCH commands

Created by a REWIND or COPY command

Sent to the terminal from another terminal

Created by execution of a program through the RUN command or through the batch facilities at the central site.

One other file – the edit file – exists whenever the user creates or updates a file through EDITOR. The edit file is a temporary work file. It has no name and does not appear on a list of the user's files.

ATTACHED FILES

The list of files associated with your user-id is obtained through the FILES command.

FILES

Files are listed under the following categories, as applicable: LOCAL, REMOTE INPUT, REMOTE EXECUTING, REMOTE OUTPUT, and REMOTE PUNCH.

REMOTE INPUT and REMOTE EXECUTING files exist only if the terminal has been used to submit a job for batch execution. Since this manual deals with job execution through the EDITOR RUN command only, such files are not described any further here.

REMOTE OUTPUT and REMOTE PUNCH files usually result from batch job execution, but they also can result from a BATCH command that sends a file from one terminal to another.

Files in the remote categories cannot be used in most INTERCOM commands until the user makes them local files. These files, and the commands to make them local files, are briefly described with the BATCH command in section 6.

The remaining file category – LOCAL FILES – encompasses almost all files used by beginners.

LOCAL files may be further categorized as follows; they are described individually below:

Attached permanent files	Special name files INPUT and OUTPUT
Connected files	Other

The most important concept to know is that a file must be a LOCAL file before it can be used at a terminal, and that LOCAL file status results automatically from most user commands.

LOCAL FILES

A local file has these characteristics:

It is immediately accessible and can be referenced in other INTERCOM commands.

It appears under the heading LOCAL FILES when the FILES command is executed.

It has been created during the current terminal session; or it is a permanent file attached at user command.

It can be made permanent with a STORE command (unless it is already an attached permanent file).

It can be used during execution of an interactive user program.

It disappears at the end of the terminal session (unless it is a permanent file).

The last item is particularly important. Unless you make a file permanent, it will be lost when LOGOUT is executed.

LOCAL FILE NAMES

Each local file must have a unique name consisting of 1 to 7 letters or digits beginning with a letter. These file names are legal:

TAPE5	NEW	ANOTHER
B1	A	PF
TEXT	Q007	OUT

The following file names should not be used unless the file has the function and characteristics associated with the name, as discussed below.

INPUT OUTPUT PUNCH

Some other file names, although legal, should be avoided while you are learning to use INTERCOM. Specifically, do not give a file the same name as any of the operating system control statements or INTERCOM commands, and do not use the reserved file names beginning with ZZZZZ.

Generally, a file name should be chosen to reflect its contents. For example, use names such as:

PROGRAM or P	For a source program
AREA	For a program calculating area of a triangle
P1 and P2 and P3	For three successive versions of a file
DATA or D or TAPE5	For data to be read by a program
NEW or FILE2	For files copied

When several persons use a terminal during a single session, it is often convenient to start or end all of your file names with your initials. Any mnemonic identifier that helps you remember the purpose of a file is useful.

Illegal names produce error message such as:

```
ILLEGAL FILE NAME

ERR – RESERVED FILE NAME
```

Examples of improper names:

333	File names must begin with a letter
AB$C	Only letters and digits can be used
ABCDEFGH	Maximum file name length is 7 characters
ZZZZZAB	Reserved file name not allowed

All local file names must be unique. An attempt to create a second file with the same name either accesses an existing file or produces a message, but does not create the file.

```
DUPLICATE FILE NAME

ERR – name ALREADY EXISTS

YOU ALREADY HAVE A FILE BY THIS NAME
```

NUMBER OF LOCAL FILES ALLOWED

The number of local files that can be associated with a terminal is limited. The specific number of files allowed is set by an installation; often 20 files are allowed. Each of the following messages indicates an attempt to create more files than you are allowed.

FILE QUOTA EXCEEDED

> Reduce the number of local files attached to your terminal. The command that caused this message has been executed.

YOU HAVE TOO MANY FILES – PLEASE RETURN SOME

> You have ignored the FILE QUOTA EXCEEDED message. Execution of other commands is inhibited until you eliminate some local files.

ERR – USER FILE LIMIT EXCEEDED

> The edit file cannot be saved or executed until the number of local files is reduced.

When any of these messages appear, it is necessary to dispose of some local files with one of the following methods. The preferred response is a user command that destroys unwanted files.

> Use the BATCH command to change the file from a LOCAL file to a REMOTE OUTPUT file at the same or another terminal. Remote output files do not affect local file quota, since they are not immediately accessible.

> Use the RETURN or DISCARD commands to evict a file. The named file ceases to exist as a local file. In addition, if DISCARD is used, the file no longer exists as a permanent file.

When you no longer need a file, eliminate it from the system with a DISCARD or RETURN command. Not only do these commands free system resources, they also reduce the number of files of which you need to keep track.

MAKING A FILE LOCAL

Any file to be used at a terminal must have LOCAL status. Unless the file name appears under the LOCAL FILES category when the FILES command is executed, the file cannot be rewound, copied, edited, or stored as a permanent file.

A file may be made local in many ways; most require explicit user naming of a file. Sometimes file names appear as an indirect result of a user command.

> A permanent file stored between terminal sessions can be attached for use by the FETCH command.

> A file created through EDITOR is given a specific name with the EDITOR command SAVE.

Referencing a file in certain commands causes a file to be created with that name if such a file does not exist. These commands are:

COPY,afile,bfile	File afile is assumed to exist; bfile is created by execution of COPY.
REWIND,cfile	File cfile is created by REWIND execution if cfile did not previously exist. Although cfile is an empty file, it does exist.

A remote OUTPUT file can be given LOCAL disposition with the BATCH command.

WHAT IS A PERMANENT FILE?

A permanent file exists from one day to the next. All local files that are not permanent cease to exist when LOGOUT occurs or a terminal session ends otherwise.

You can make a permanent file of any local file unless it is already permanent.

Three commands reference permanent files:

STORE	Makes a local file permanent
FETCH	Retrieves a permanent file from storage
DISCARD	Destroys a permanent file

For the most part, an attached permanent file can be treated as any other local file. It can be transferred to the edit file for listing, executed by the RUN command, or copied to another file; however, an attached permanent file cannot be overwritten or otherwise have its contents changed with INTERCOM or EDITOR commands.

On a list of local files, attached permanent file names are preceded by an asterisk which is not part of the name.

The STORE, FETCH, and DISCARD commands are simple-to-use versions of the operating system control statements CATALOG, ATTACH, and PURGE. These statements offer additional features of privacy, naming, and modifying files, but their use requires more sophisticated knowledge than is needed for the most commonly used permanent file features.

STORE is adequate for retaining files from one day to the next. It offers the user a feature that CATALOG does not have: specifically, if the file to be made permanent is not on a mass storage device that can hold permanent files, STORE will copy the file to such a device for the user.

A permanent file remains in the system until:

It is referenced in a DISCARD command; or

The installation purges it after a certain retention period.

When a file is made permanent with STORE, it acquires an expiration date (creation date plus retention period). Depending on installation policy, your permanent files may be purged without advance information on the date they expire. Some installations transfer expired files to an archive tape, which may cause a delay in associating the file with your terminal. The AUDIT utility shows the expiration date for each of your files. Although permanent files serve a need in preserving files, they tie up system resources and should not be used needlessly.

WHAT IS A CONNECTED FILE?

Connected files are associated with the keyboard or display of the terminal.

> If a connected file is referenced in a program write operation, a line is displayed immediately.
>
> If a connected filé is referenced in a program read operation, a line must be entered at the keyboard for immediate use.

Two commands are used for connected files:

CONNECT	Connects a file
DISCONT	Disconnects a connected file

Files INPUT and OUTPUT are always connected by EDITOR when a program is executed through the RUN command of EDITOR.

On a list of local files, connected file names are preceded by a dollar sign. The $ is not part of the name.

Information written to a connected file while it is connected goes to the executing program for use or to the terminal for immediate display. No mass storage copy exists for any information written to a connected file.

Before a file is connected, and after it is disconnected, information written to the file exists on mass storage, the same as for any other local file. Existing mass storage information becomes inaccessible while the file is connected, but it is not destroyed and can be used after a disconnect.

An attempt to transfer a connected file to the edit file work area for a LIST produces the message: ERR-FILE name CONNECTED TO TERMINAL.

Connected files cannot be referenced in a RETURN or DISCARD command until they are first disconnected. Further, they cannot be referenced in a REWIND command since terminal information cannot be rewound.

SPECIAL FILE NAMES

Generally, the user can choose any sequence of letters for a file name. Several file names, however, have special meaning throughout the computer system. These names should not be used indiscriminately. Two of these names are INPUT and OUTPUT.

The system assumes any file with the name OUTPUT contains information to be printed. Both system routines and a user program can write to OUTPUT. For example, the results of the AUDIT utility, which produces permanent file status information, is written to OUTPUT: execution of AUDIT causes an OUTPUT file to be created if it does not already exist.

Since the user at a terminal does not have a printer available, information that would otherwise appear on printer output appears at the terminal. When a program is executed through the RUN command of EDITOR, for instance, the system always creates a file with the name OUTPUT; it is connected to the terminal so that any program print operations are displayed immediately.

Any file with the name INPUT also has special meaning throughout the system. For example, in a job deck submitted at the central site, INPUT refers to the deck itself. The COBOL compiler assumes that the program to be compiled exists on the file with the name INPUT.

If a file named INPUT or OUTPUT is a local file, the system uses it in its customary way, even if you did not intend such use. Consequently, avoid these names until you are sure of the implications.

Using the RUN command of EDITOR causes both INPUT and OUTPUT to exist as connected files. Examples in section 5 explain their use.

THE EDITOR OF INTERCOM

EDITOR is the file creating and editing facility of INTERCOM. It is a utility program that must be called by user command.

At most terminal sessions, EDITOR is one of the first commands entered after LOGIN. EDITOR makes it possible to:

Enter a program line by line, equivalent to keypunching it card by card

Correct any errors in the program

Execute the program and receive output

Save the corrected program for future use

When EDITOR has been entered, the terminal is considered to be in EDITOR mode. You can distinguish EDITOR mode by the system response to a user entry. Two dots appear to the left of a new line in EDITOR mode as shown below:

```
..
  ↑
  You can begin entering command here
```

Once in EDITOR mode, you can access a set of commands which are applicable only in EDITOR mode. They are summarized in table 3-2.

TABLE 3-2. VALID EDITOR COMMANDS

Command	Description
ADD	Inserts new lines into edit file
BYE	Exits from EDITOR
CREATE	Begins new edit file creation
DELETE	Removes line from edit file
EDIT	Copies local file into the edit file
FORMAT	Specifies edit file format
LIST	Displays edit file contents
RESEQ	Renumbers lines in edit file
RUN	Compiles and executes program
SAVE	Copies edit file to a local file
linenum=text	Inserts single line into edit file
/oldtext/=/newtext/	Changes text string in edit file

Two additional entries are valid only in EDITOR:

=	Exits from ADD or CREATE in which a file is begin created line by line
*EOR and *EOF	Writes file terminators to edit file

When the character strings *EOR and *EOF begin a line in the edit file, they are converted to end-of-record and/or end-of-file indicators recognizable by other parts of the operating system when the edit file is made a local file by the SAVE command.

If you try to use a command listed in table 3-2 before calling EDITOR, the command is interpreted as a call for loading a user file by that name, and the NO SUCH PROGRAM CALL NAME diagnostic appears.

Once EDITOR has been called, almost all INTERCOM commands can be entered as well as commands applicable only to EDITOR. Those few commands not possible from EDITOR include LOGOUT. Do not be concerned about other commands and whether or not you can use them from EDITOR. Assume all commands are possible and use them as you wish. Any improper command is diagnosed but not executed; and this message appears:

```
COMMAND NOT ALLOWED FROM EDITOR
```

EDITOR COMMAND SYNTAX

EDITOR commands are similar to INTERCOM commands in that:

The RETURN key signals the end of a command.

The physical line on which a command or part of a command appears is not significant.

A separator must appear between parameters.

The syntax differs from INTERCOM commands in that:

A space may be used to separate parameters.

One or more spaces or commas between parameters is considered to be a single separator.

Both command names and parameters may be abbreviated.

Commands must not end with a period.

As an illustration, consider the use of an EDITOR command SAVE and the INTERCOM command STORE. SAVE is used to copy a file created through EDITOR and make it a named local file; STORE preserves the file for the next terminal session.

SAVE format for naming as file MEW

```
SAVE,MEW,NOSEQ
↑    ↑   ↑
|    |   └─ SAVE optional parameter which removes EDITOR line numbers
|    |      before copying the file
|    |
|    └─ user assigned file name
|
└─ EDITOR command
```

STORE format is:

```
STORE,MEW,GULL
      ↑   ↑
      |   └─ owner identifier
      |
      └─same name as on SAVE
```

The SAVE command can be entered several ways (where $_\wedge$ represents a space bar entry):

SAVE,MEW,NOSEQ   SAVE,$_{\wedge\wedge\wedge\wedge}MEW_\wedge$NOSEQ

SAVE$_\wedge$MEW$_\wedge$NOSEQ   S$_\wedge$MEW$_\wedge$N

The STORE command must be entered this way:

```
STORE,MEW,GULL
```

Until you are sure what is or is not an EDITOR command, use a single comma between parameters and do not end with a period. Since the required syntax for INTERCOM commands is acceptable for EDITOR commands, you always will have an acceptable format.

ABBREVIATED COMMANDS

Both the command name and parameters of EDITOR commands can be abbreviated. The minimum abbreviation is as many letters as necessary to identify a parameter uniquely.

CREATE can be abbreviated C

SAVE can be abbreviated S

LIST can be abbreviated L

Two commands require a two letter minimum abbreviation. The letter R could be intended for RUN or RESEQ.† so abbreviations are:

RU for RUN

RE for RESEQ

The rule for abbreviations of command and parameter names is that you can use as many letters as you care to, as long as the string can be uniquely identified as the proper name.

CREATE can be abbreviated these ways, in addition to C:

CR   CRE   CREA   CREAT

SAVE can be abbreviated:

S   SA   SAV

Abbreviations are not used in this manual so that you will be reminded of the purpose for each command or parameter. Once you are familiar with EDITOR use, substitute abbreviations to reduce the possibility of input typing errors.

†In actual operation, R is interpreted as an abbreviation for READ, a command that is possible only if a card reader is attached to your terminal; thus the COMMAND NOT ALLOWED FROM EDITOR diagnostic appears if you use it instead of RU or RE.

THE EDIT FILE

The edit file is a temporary work area used to accumulate lines of a file being created or to hold a file that will be changed.

When the user types CREATE, an edit file is begun. EDITOR accumulates input characters as a series of lines; RETURN key input signals the end of each line. Edit file construction is terminated when the user enters an equals sign. All information entered remains in the edit file until the user calls for line deletion or change or for edit file destruction.

Only one edit file can exist at any given time. You have a choice of creating an edit file with the CREATE command or transferring an existing file to the edit file.

Any existing file to be updated by adding new lines or changing characters in old lines must first be moved to the edit file with the EDIT command. No file updating is possible unless the file exists in the edit file work area.

To protect the user from inadvertently destroying the edit file, EDITOR issues the following warning message after a BYE, EDIT, or CREATE command (unless the edit file has been referenced in a SAVE command):

```
WARNING - EDIT FILE NOT SAVED
```

The command is not executed at this point. When the message appears, the user can issue a SAVE, re-issue the BYE, EDIT, or CREATE command, or enter any other legal command.

> SAVE preserves the current edit file by copying it to a local file; thereby freeing the edit file for use in creating another file. After issuing the SAVE, the previously rejected BYE, EDIT, or CREATE must be re-entered.
>
> BYE causes an exit from EDITOR.
>
> EDIT destroys the current edit file by writing the named file over the file existing in the work area.
>
> CREATE destroys the current edit file by beginning a new file in the work area.

When you leave EDITOR and return without logging out, the following message appears:

```
YOU HAVE AN EXISTING EDIT FILE
```

At this point you have the option to save the edit file (if not previously saved) or to enter any other legal command.

A file in the edit file area must have a line number associated with every individual statement. These numbers are necessary for EDITOR to identify the specific changes you want to make. Numbers are assigned automatically by CREATE or by using the SEQ parameter of the EDIT command.

The following sections show many examples of creating and modifying files through EDITOR.

FILE CREATING AND UPDATING 4

EDITOR commands allow a file to be created or updated line by line. To call EDITOR, enter the characters EDITOR for an INTERCOM command.

```
    COMMAND – EDITOR
```

Don't forget to press RETURN to end this and all other entries.

In EDITOR mode, two dots appear at the left to show the system has completed execution of the previous entry and is ready to accept another command.

```
    ••
```

EDITOR remains available until you enter the command BYE.

Follow the text from beginning to end in this section as COBOL programs are created and executed. Once you are familiar with each step for entering and updating a file, you can turn to section 6 for more details and common uses for a given command.

FILE CREATION AND EXECUTION

The following discussion introduces the use of these EDITOR commands:

CREATE	FORMAT	LIST	RUN

The minimum parameters needed for a task are emphasized. Optional parameters are used only when they are pertinent to the task at hand.

CREATE COMMAND FUNDAMENTALS

With the CREATE command, a program can be entered statement by statement. Each statement entered is assigned a line number that has meaning only to EDITOR. The numbers make it possible to call out any individual line for deletion, correction, or other manipulation without affecting other lines.

CREATE signals EDITOR to begin a new file in the temporary work area called the edit file. Since only one edit file can exist, creation of a new file destroys the present edit file contents. Consequently, EDITOR may display a warning message before overwriting the edit file; if so, you must re-enter the CREATE command after considering whether to save the existing file first.

When CREATE is called, EDITOR responds with the number 100 and an equals sign:

```
••CREATE
   100=
```

You may enter as many as 72 characters before signaling the end of the line with a RETURN key:

```
••CREATE
   100=USER ENTERS A STATEMENT
```

EDITOR prompts input of another line by displaying a line number with a value 10 greater than the last line number:

```
••CREATE
   100=USER ENTERS A STATEMENT
   110=
```

The user continues adding lines when prompted by a line number, terminating each line with a RETURN key.

You can specify the starting line number and line increment as follows:

```
••CREATE,1,5
     1=USER ENTERS A STATEMENT
     6=
```

When all your statements have been entered, exit from CREATE by entering an equals sign as the first and only character in a line. The equals sign does not become part of the file.

```
••CREATE
   100=STATEMENT. END OF LINE 100 IS SIGNALED BY RETURN KEY
   110=SPACES ARE SIGNIFICANT IN LINES ENTERED
   120=LINES NEED NOT BE THE SAME LENGTH
   130=SPECIAL CHARACTERS SUCH AS ( ) / # $ CAN BE ENTERED
   140=LINE 150 WILL BE THE LAST LINE OF TEXT INPUT HERE
   150=ONLY AN EQUALS SIGN (=) WILL BE ENTERED FOR LINE 160
   160==
••
```

Figure 4-1. Fundamental CREATE Use

The statements in figure 4-1 present some new information about CREATE use: letters, digits, and special characters can be part of a line.

Most often you will use CREATE to enter lines of a program that has a purpose, such as calculation of a loan amortization. But as figure 4-1 shows, the file created through EDITOR need not be a program; it could be a file of 999 test scores to be averaged, or a single line of an input statement to be read by a COBOL program. All CREATE options are explained in detail in section 6.

LIST COMMAND FUNDAMENTALS

The original lines entered remain in the edit file until they are destroyed. To examine the edit file, use the LIST command (LIST can be used only for the edit file) as shown in figure 4–2.

```
..LIST,ALL
   100=STATEMENT. END OF LINE 100 IS SIGNALED BY RETURN KEY
   110=SPACES ARE SIGNIFICANT IN LINES ENTERED
   120=LINES NEED NOT BE THE SAME LENGTH
   130=SPECIAL CHARACTERS SUCH AS , . / * $ CAN BE ENTERED
   140=LINE 150 WILL BE THE LAST LINE OF TEXT INPUT HERE
   150=ONLY AN EQUALS SIGN (=) WILL BE ENTERED FOR LINE 160
..
```

Figure 4–2. Listing Edit File

Notice the equals sign, entered as line 160 in example 4-1 to exit from CREATE, is not part of the file itself.

The ALL parameter caused the entire file to be displayed. Other LIST options can:

Display a single line of the edit file.

Display a range of lines.

Display lines without listing line numbers.

Search and display all lines containing a particular character string.

Search and display all lines in a particular range which contain a particular character string.

Restrict search for a character string to particular positions in each line.

All LIST options are explained in detail in section 6 and illustrated later in this section.

FORMAT COMMAND FUNDAMENTALS

Now that you can enter a series of lines with miscellaneous data, let's see what may be different about entering a meaningful program.

The most obvious difference is that characters entered for each line normally would not begin in the first character position immediately after the line number displayed by EDITOR.

A program to be executed must be entered in a format expected by the language compiler. Consider, for example, a program punched on standard cards with 80 columns. All the language compilers accept information punched in columns 1 through 72. Any information in columns 73-80 is listed but ignored during compilation.

The COBOL compiler attaches special meaning to some of the columns.

Columns 1-6	Numbers in this position are line sequence numbers.
Column 7	A hypen in this position indicates a continuation of the statement in the previous line. A slash or asterisk indicates that the remainder of this line is treated as comment.
Column 8	Statements must begin in this position or beyond.

The column in which information begins affects the interpretation of that information. For example, a paragraph name must begin in column 8; the first sentence of a paragraph must begin in column 12.

Figure 4-3 is a complete COBOL program that prints three lines during execution.

```
100=;ID DIVISION.
110=;PROGRAM-ID. PRINTIT.
120=;ENVIRONMENT DIVISION.
130=;CONFIGURATION SECTION.
140=;SPECIAL-NAMES. TERMINAL IS HERE.
150=;DATA DIVISION.
160=;PROCEDURE DIVISION.
170=;FIRST-PARA.
180=;;DISPLAY "A SEMI-COLON CAUSES A SKIP TO COL 8" UPON HERE.
190=;;DISPLAY "THE NEXT SEMI-COLON SKIPS TO COL 12" UPON HERE.
200=;;DISPLAY "IF YOU ARE USING FORMAT,COBOL." UPON HERE.
210=;;STOP RUN.
220==
..
```

Figure 4-3. Program PRINTIT

Aligning COBOL statements in position could have been accomplished by spacing to position before entering the first character. It's much easier to take advantage of the format and tab capabilities of EDITOR.

To establish a format suitable for a COBOL program, enter this command before calling CREATE.

```
. . FORMAT,COBOL
```

When this command is accepted, EDITOR assumes the edit file has these characteristics:

Each line is limited to 72 characters, excluding the EDITOR assigned line numbers.

The tab character is a semicolon (;).

Tab stop positions are 8, 12, 16, 20, 24.

The tab character acts similarly to the tab key on a typewriter or a skip key on a keypunch to allow input to be positioned on a line. Pressing the RETURN key at a terminal moves the carriage to the far left, and the next character entered occupies the first character position of the line. Pressing the tab key causes the next character entered to occupy the same character position as the tab stop.

The tab for EDITOR, unlike that of a typewriter, is a character and not a particular key. Further, the character that signals a tab can be changed by the user.

When the tab character is entered, EDITOR adds blanks to move its internal position marker to the next tab position. Although the terminal carriage does not move in response to tab use, the position marker internal to EDITOR is changed. When the line entered with a tab character is displayed at a terminal, the carriage moves to correctly position characters in a line.

For example, assume the semicolon is the tab character and the first tab stop is in position 8. Under CREATE, the user enters for line 310:

```
310=;PARA-20
    ↑↑
    ||__position 2
    |
    position 1
```

When the file containing that line is displayed, characters are aligned:

```
310=          PARA-20
              ↑
              position 8
```

The line is passed to the compiler with PARA-20 in columns 8-14 and the line number in columns 73-78.

The tab character does not appear as part of the user data. It simply signals a change in the EDITOR internal line position marker.

Using the tab character, the above example could be entered as shown in figure 4-4.

```
100=;ID DIVISION.
110=           PROGRAM-ID. PRINTIT.
120=   ;ENVIRONMENT DIVISION.
130=;CONFIGURATION SECTION.
140=    ;SPECIAL-NAMES. TERMINAL IS HERE.
150=           DATA DIVISION.
160=;PROCEDURE DIVISION.
170=;FIRST-PARA.
180=          ;DISPLAY "A SEMI-COLON CAUSES A SKIP TO COL , " UPON HERE.
190=                 DISPLAY "THE NEXT SEMI-COLON SKIPS TO COL 12" UPON HERE.
200=;;DISPLAY "IF YOU ARE USING FORMAT,COBOL. "UPON HERE.
210=;;STOP RUN.
220==
```

Figure 4-4. Tab Character Use with CREATE

Notice, the tab character need not be used in all lines nor used at any particular time. Spaces before the tab character can be entered. The tab character is effective in any position before the tab stop.

The files resulting from lines entered as in figure 4-3 and 4-4 are identical.

The contents of the file created in figure 4-4 can be displayed by LIST (refer to figure 4-5). Adding the SUP parameter to LIST suppresses the line numbers; it does not otherwise affect contents.

```
••LIST,ALL,SUP
       ID DIVISION.
       PROGRAM-ID. PRINTIT.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES. TERMINAL IS HERE.
       DATA DIVISION.
       PROCEDURE DIVISION.
       FIRST-PARA.
           DISPLAY "A SEMI-COLON CAUSES A SKIP TO COL 8" UPON HERE.
           DISPLAY "THE NEXT SEMI-COLON SKIPS TO COL 12" UPON HERE.
           DISPLAY "IF YOU ARE USING FORMAT,COBOL." UPON HERE.
           STOP RUN.
••
```

Figure 4-5. LIST of Program Entered in Figure 4-4.

Neither the tab character nor the = used to exit from CREATE mode is part of the file created. The lines are formatted as required by the COBOL compiler.

In line 130, figure 4-4, the word CONFIGURATION begins in position 8.

In line 150, the word DATA begins in position 8.

Although line 200, figure 4-4, is entered with 11 characters in positions 1-11, the resulting file line occupies 20 characters. The tab characters, semicolons, cause the character S to be in position 12. The character string STOP RUN. occupies positions 12-20.

Entry at terminal:

```
                entry position 11 of line 200
                ↓
200=;;STOP RUN.
    ↑
    logical position 12 of final line 200
```

Line passed to compiler:

```
ƀƀƀƀƀƀƀƀƀƀƀSTOPƀRUN.
           ↑
           position 12
```

The final internal line length, as opposed to the number of characters entered at the terminal, is significant. The FORMAT specification for COBOL limits line length of the file being created to 72 characters.

Line 170 of figure 4-4 has been entered with 62 characters (a semicolon, 56 characters, and 5 leading spaces). When this line is formatted by EDITOR, it will occupy 67 character positions, with the word DISPLAY beginning in position 12.

Let's see what happens if more than 72 characters are entered when one of the 72 is the tab character (use the line feed to enter more than a single Teletype line can hold).

```
..130=;DISPLAYb
              ↑
         line feed
 "A SEMICOLON CAUSES A SKIP TO COLUMN 8 WHEN ADD OR CREATE
                                        MODE IS BEING USED"b
                                                          ↑↑
                                 entry position 93 of line 130 |
                                                         RETURN

MOD... TRUNCATED FROM LONG LINE
```

EDITOR does not accept more than 72 characters while the FORMAT,COBOL command is in effect. The tab character is replaced by seven spaces, in this case, thus expanding line length to 79 characters. The first 3 characters of the diagnostic message show where line truncation began.

This example also illustrates that the physical Teletype line on which an edit file line is entered does not affect the meaning of the EDITOR line. An edit file line is not considered complete until a RETURN key is pressed; a LINE FEED entered after the word DISPLAY in line number 130 did not start either an edit file or a COBOL program line. The LINE FEED is interpreted as a continuation signal.

If, as in line 130, a blank is required between the last word of the first line and the first word of the second physical line, that blank must be entered at the beginning of the second line, or at the end of the first line (before the line feed).

```
130=     DISPLAY "A SEMI-COLON CAUSES A SKIP TO COLUMN 8 WHEN ADD OR
CREATE MODE IS BEING USED."
↑                          ↑
blank                    RETURN                                  line feed
```

Without the blank before the C in CREATE, printed output would appear; ORCREATE.

RUN COMMAND FUNDAMENTALS

Now that the edit file contains a valid COBOL program, the program can be compiled and executed through EDITOR by using the RUN command. The RUN command must specify the language in which the program is written. The RUN command is then:

```
..RUN,COBOL
```

Since no file name is given, RUN works with the current edit file.

EDITOR first makes a copy of the edit file, then passes it to the COBOL compiler. At the end of compilation, a status message is displayed showing how long the central processing unit at the central site was in use:

```
    .200 CP SECONDS COMPILATION TIME
```

Processing of the RUN command continues by loading the compiled program (the machine code translation of your program) into central memory and starting execution.

The program PRINTIT, in figure 4-4, contained three DISPLAY statements. Successful execution should show these three lines on a COBOL-provided connected file called TERMINL.

As RUN continues, you will see the DISPLAY statements appear at the terminal as shown in figure 4-6.

```
 COMPILING PRINTIT
  000 E AND     T/U DIAGNOSTICS ISSUED
     053277B SCM USED
       .151 CP SECONDS COMPILATION TIME
 END COBOL
A SEMI-COLON CAUSES A SKIP TO COL 8
THE NEXT SEMI-COLON SKIPS TO COL 12
IF YOU ARE USING FORMAT,COBOL.
..
```

Figure 4-6. Output from PROGRAM PRINTIT

In addition to the three lines output because of the DISPLAY statements, the words END COBOL appear. END COBOL is displayed by the compiler at the end of the compilation. Unless your program contains statements to display results, there will be no indication of successful execution other than the EDITOR response (. .).

Since execution of program PRINTIT displayed three lines destined for a connected file, you would expect to find that file (TERMINL) in your list of files. To see what files you do have, use the FILES command.

```
    . . FILES
```

FILES provides a list of all file names associated with your user name as shown in figure 4-7.

```
..FILES
--LOCAL FILES--
  $INPUT       $OUTPUT       LGO        $TERMINL
..
```

Figure 4-7. Files Generated by RUN Command

Notice that TERMINL is listed as a connected file. There are three other files that RUN produces.

LGO	Contains the compiled program which was created, loaded, and executed during RUN. It still contains the machine code, and can be executed again.
INPUT	Has a $ preceding the name to indicate it is a connected file. The file was created and connected in anticipation of use during PRINTIT execution. As it happens, PRINTIT did not reference a file by the name INPUT, and the file was not used.
OUTPUT	Also is a connected file created by RUN in anticipation of use in PRINTIT. If any program statement writes to a file named OUTPUT, the line will be displayed immediately at the terminal. After execution, data is not on the file. Again, PRINTIT did not reference a file by the name of OUTPUT, and the file was not used.

These three files always are created by RUN execution. As program PRINTIT shows, however, it is not necessary for the program to use files INPUT and OUTPUT.

OUTPUT would have been used, rather than TERMINL, if the COBOL program had not used the SPECIAL-NAMES feature. In that situation, the DISPLAY verb would have caused the lines to be written to OUTPUT rather than HERE, which is in fact TERMINL. Since RUN automatically connects OUTPUT, this technique would have worked.

Section 5 contains examples of connected file use.

RUN execution does not affect the current contents of the edit file. If you want to rerun the program, simply re-enter the RUN command. Files OUTPUT and LGO created by the previous RUN execution are logically destroyed with the second RUN execution: first these files are positioned to their beginning, and then during PRINTIT compilation and execution, they are written over.

The RUN command has several optional parameters in addition to the required parameter that specifies the language in which the program is written. These options allow, as discussed in section 6:

1. Compilation and execution of a program that is not currently in the edit file.
2. Suppression of informative or nonfatal error diagnostics.
3. Compilation, but not execution, of a program.

This last option is particularly useful when you are first learning COBOL, since it allows compilation errors to be detected without wasting computer resources in an attempt to execute an erroneous program.

SUMMARY OF PROGRAM ENTRY AND EXECUTION

Call EDITOR if two dots do not appear at the left of a line when RETURN is pressed.

```
COMMAND-EDITOR
••
```

Call for formatting according to COBOL: tabs at positions 8, 12, 16, 20, 24 by semicolon, 72 character line length.

```
..FORMAT,COBOL
..
```

Call CREATE. Optional parameters can specify starting line numbers and increment between lines.

```
..CREATE,50,2
..
```

Enter each line in file after EDITOR prompting by display of a line number. Use tab character for skip to position 8. Terminate each line by pressing RETURN key.

After all program lines have been entered, type a single = to terminate CREATE mode.

```
60==
```

Verify that the data has been entered correctly, by listing the edit file.

```
..LIST,ALL
```

Call RUN to execute program:

```
..RUN,COBOL
.
.
.
END COBOL
```

FILE PRESERVATION AND ELIMINATION

Once a file has been created through EDITOR, it can be executed through the RUN command. To put the file aside while you are creating another file, you must make the edit file into a local file. Then, if the local file is to be preserved until another terminal session, it must be made into a permanent file. Alternately, if a file has served its purpose and is no longer needed, it should be eliminated to conserve system resources.

The following description introduces use of these commands:

SAVE	RETURN	STORE	FETCH	DISCARD

SAVE COMMAND FUNDAMENTALS

A file entered under the EDITOR command CREATE exists in a temporary work area called the edit file. The fact that only one edit file can exist does not imply that you can create or use only a single file. Rather, you must take steps to preserve the present edit file before creating or updating another file in the edit file work area.

As long as a file resides in the edit work area, it has no name and is destroyed when a new file is constructed through CREATE or EDIT.

If you are following this section of the manual from beginning to end and studying examples as you read, you will see that the edit file now has PROGRAM PRINTIT, as listed in figure 4-5.

To make the current contents of the edit file available for use in other commands, use the SAVE command to copy the edit file and give it a name.

```
. . SAVE,filename
```

Existence of the new file can be verified by the FILES command as shown in figure 4-8.

```
..FILES
NONE
..SAVE,DOVE
..FILES
--LOCAL FILES--
   DOVE
..
```

Figure 4-8. Saving the Edit File

Saving the file does not affect the original edit file contents. It merely makes a copy. You can make another copy with another SAVE command. Let's make four copies, in all, and use the FILES command to check that they exist.

```
..FILES
--LOCAL FILES--
   DOVE
..SAVE,DOVE2
..SAVE,DOVE3
..SAVE,DOVE4
..FILES
--LOCAL FILES--
   DOVE2        DOVE3        DOVE4        DOVE
..
```

The only required parameter of SAVE is the name to be given to the local file. A file name can be any combination of 1 to 7 letters or digits beginning with a letter; it must be different from all other names listed by the FILES command under the LOCAL FILES heading.

Optional parameters of SAVE can:

Save a single line of the edit file.

Save a range of lines.

Save lines without saving line numbers.

Search and save all lines containing a particular character string.

Search and save all lines in a particular range which contain a particular character string.

Restrict search for a character string to particular positions in each line.

Replace the current contents of a local file by the saved lines.

Add the saved lines to the current contents of a local file.

If the options of the SAVE command seem familiar, look at the LIST command options presented previously and note they are similar. Four EDITOR commands use many of the same optional parameters. When you learn the options of LIST, you will know all but one option of the text replacement command /oldtext/=/newtext/ and the DELETE command, and all but two optional parameters of SAVE.

RETURN COMMAND FUNDAMENTALS

Since four copies of PROGRAM PRINTIT serve no purpose other than to illustrate that they can be made, eliminate 3 copies.

Use the RETURN command to eliminate unwanted files:

```
. . RETURN,filename
```

More than one file can be eliminated by a command in the format:

```
. . RETURN,filename1,filename2,filename3 . . .
```

RETURN is a system command; it is not an EDITOR command. This means simply that RETURN can be executed even if EDITOR is not in use. Any command that is identified as an EDITOR command cannot be used until after EDITOR is called.

For example:

```
                         COMMAND-EDITOR          )
                         . .                     |
                             editor file creating  > most INTERCOM and system commands can be
                             or updating statements| used here
                         . . BYE                 )
                       ( COMMAND-RETURN,A        )
all INTERCOM           | COMMAND-                |
and system commands   <      commands go here     > EDITOR commands cannot be used here
can be used here       |       .                 |
                       |       .                 |
                       ( COMMAND-LOGOUT          )
```

Valid EDITOR commands used when EDITOR has not been called normally result in a diagnostic message beginning:

```
                                          EDITOR command name
                                                  ↓
    NO SUCH PROGRAM CALL NAME   -   LIST
```

If EDITOR has not been called, an EDITOR command cannot be recognized. The system assumes a command such as LIST to be a call for loading and executing a file by the name of LIST; as no file with this name was found to be associated with the terminal, a loader error resulted.

Now, to resume discussion of the RETURN command, figure 4-9 shows use of RETURN and the corresponding change that occurs in the list of files associated with the terminal.

```
..RETURN,DOVE2,DOVE3,DOVE4
..FILES
--LOCAL FILES--
  DOVE
..
```

Figure 4-9. Eliminating Unneeded Files

RETURN can be used for any unconnected local file except one with the name INPUT. (Refer to descriptions of the DISCARD and the RENAME option of the BATCH command in section 6 for means of eliminating this file.) As long as you are executing programs only through the RUN command of EDITOR, don't worry about more than one program using these special name files or about the same program using them several times.

Experienced programmers calling the compilers themselves must be concerned with file positioning. But for beginners, RUN command execution handles these files to produce the results desired without additional user entries.

Normally, you will use RETURN to destroy any file you no longer need; but don't forget that once a file is referenced in RETURN, it is gone and cannot be recovered without duplicating the process that created it. (The exception, of course, is a file made permanent by a STORE command; only a DISCARD command will destroy the permanent copy of the file. RETURN of a permanent file merely disassociates the file from the terminal.) If you are passing the terminal to someone else, return all your files so the next person may start clean.

STORE AND FETCH COMMAND FUNDAMENTALS

Any file referenced in a SAVE command, as file DOVE was in figure 4-8, exists as a local file. Local files are destroyed when the LOGOUT command is executed.

To preserve a local file between terminal sessions, it must be made a permanent file with the STORE command.

```
. . STORE,filename,owner
```

When STORE is executed, the file attains a special status in the system. The file remains in the system from day to day (barring unforeseen circumstances) until the user purges the file from the system through a DISCARD command. Attaching the file to the terminal through the FETCH command and using it during the terminal session does not affect its permanent status. After a certain number of days, called the retention period, permanent files may be purged by the central site operator. Each central site installation has a different policy, so check with your instructor or analyst about retention periods.

Remember that permanent files tie up system resources. Do not create them needlessly.

When a file has been made permanent, it appears on the list of local files with an asterisk before the name.

```
••STORE,DOVE,TURTLE
••FILES   |    └──────my name; I am S. Turtle
--LOCAL FILES--───local file name
   *DOVE
••
```

The asterisk is not part of the file name; it is merely a status indicator.

If you LOGOUT or execute a RETURN for an attached permanent file, the file still exists. At a later time, you can again make it a local file by using the FETCH command. To destroy the file completely, you must use the DISCARD command.

Whether or not you can use permanent files, and the specific format of the commands to use, depends on the policy of your instructor or installation. The owner identification may not be required; the formats shown below are those of the standard system, which are not necessarily applicable to all systems.

In the STORE format above, the two required parameters were:

local file name of 1-7 letters or digits which becomes the permanent file name (permname parameter in examples below)

owner identification of 1-9 letters or digits

These two parameters uniquely identify the file at the central site. No other file can be made permanent with the same name and owner. These same two parameters must be used on the other two permanent file commands, FETCH and DISCARD.

When a file is made permanent, it still exists as a local file at your terminal. RETURN eliminates the name from the list of local files.

To attach a stored permanent file for use at another terminal session, use the FETCH command:

```
. . FETCH,permname,owner
```

The two parameters must duplicate the parameters used to STORE the file. If you have forgotten the particular combination of characters used in naming your file, use the AUDIT utility, described in section 6, to obtain a list of all permanent files stored with your owner identification.

A permanent file is eliminated from the system by DISCARD. If the file has been made local with the FETCH command, DISCARD format is:

```
. . DISCARD,permname
```

If the file is stored at the central site but has not been made a local file, format is:

```
. . DISCARD,permname,owner
```

In either instance, DISCARD destroys the permanent file and any local file copy that may exist.

The STORE, FETCH, and DISCARD commands are INTERCOM commands and can be used any time, even when EDITOR has not been called.

Figure 4-10 shows a program that was stored as a permanent file on one day and attached and used again the next day.

```
LOGIN,NAME,PASS

      (LOGIN MESSAGES APPEAR HERE)

COMMAND-EDITOR
..CREATE

   (ENTER PROGRAM HERE)

..SAVE,PROG
..STORE,PROG,TURTLE
..BYE
COMMAND-LOGOUT

(LOGOUT MESSAGES APPEAR HERE)

LOGIN,NAME,PASS,SUP

COMMAND-FETCH,PROG,TURTLE
COMMAND-EDITOR
..EDIT,PROG
..RUN,COBOL
```

Figure 4-10. STORE and FETCH Use for Permanent Files

FILE EDITING

An existing file can be changed in several ways:

New lines can be added with the ADD command.

A single line can be added or rewritten with the linenum=text command.

Existing lines can be deleted with the DELETE command.

A character string in a line in the file can be replaced with another string of characters.

The changes made to a file may be required to add new information or to alter program logic, as well as to correct errors in existing information. Changing an existing file, whether the change affects an entire line or a single character in a line, is considered file editing or updating.

The following description introduces these EDITOR commands that update a file:

/oldtext/=/newtext/	EDIT	linenum=text	ADD	DELETE

New options of CREATE, LIST, and RUN are also included.

Changes cannot be made unless a file resides in the edit file work area. The current edit file contents always can be updated. The first editing command, /oldtext/=/newtext/, is illustrated in figure 4-11.

CREATE COMMAND WITH SUPPRESS

When you call CREATE, you can choose to enter your text by either of the following procedures:

1. After a line number appears, enter a line, press RETURN, and wait until another line number appears before entering the next line.

2. Enter a line, press RETURN, and enter another line.

To suppress line number prompting, enter CREATE with a SUP parameter.

. . CREATE,SUP	abbreviated C, S
ENTER LINES	(system response when you can enter new lines)

The typing element is at the far left. Enter each program statement, using the semicolon as a tab character. The = sign must be entered as the last line to terminate CREATE.

In figure 4-11 below, we have a very inaccurate typist.

```
..CREATE,SUP
ENTER LINES
;WORKING-STORAGE SECTION.
;01;HEAD.
;;03;FILER PICTJRE 9 VALUE 1.
;;03;FILLER PICTURE X(9) VALFU "JOB CLASS".
;;03;  FILLER PICTJRE X(17) VALUF SPACES.
;;03;FILLER PICTURE X(4) VALUE "NAME".
;03;FILLER PICTURE X(105) VALUE SPACES.
=
..LIST,ALL
       100= WORKING-STORAGE SECTION.
       110= 01  HEAD.
       120=       03  FILER PICTJRE 9 VALUE 1.
       130=       03  FILLER PICTURF X(9) VALEU "JOB CLASS".
       140=       03    FILLER PICTJRE X(17) VALUE SPACES.
       150=       03  FILLER PICTURF X(4) VALUE "NAME".
       160= 03  FILLER PICTURE X(105) VALUE SPACES.
```

Figure 4-11. CREATE,SUP Entry

/oldtext/=/newtext/ COMMAND FUNDAMENTALS

EDITOR contains an updating command that affects characters within a line or several lines in the edit file. It can be used to correct typing input errors, as illustrated below, or to change the meaning of a particular line.

Look at the program entered in figure 4-11, it is obvious that it does not meet compiler standards:

FILLER and PICTURE are misspelled in line 120

VALUE is misspelled in line 130

PICTURE is misspelled in line 140

Line 160 does not begin in position 12

To correct an error without reentering the entire erroneous line, use the text replacement command. The command specifies which characters are to be replaced by other characters. Both the original character string and the replacement string must be delimited by any characters that are not part of the string (however, the delimiters must be the same for any string). The minimum command format is:

/oldtext/=/newtext/,range

The range parameter is required to identify the portion of the file EDITOR is to search for the character string /oldtext/. It can have any of the formats valid for the range parameter of the DELETE, SAVE, or LIST commands.

Once the character string identified by /oldtext/ is found, the characters are replaced with the string identified by /newtext/. Text strings need not be the same length:

/oldtext/ may be 1–20 characters

/newtext/ may be 0–20 characters

To correct line 130 only in figure 4–11, specify the bad characters and the replacement characters:

```
../EU/=/UE/,130
  1 CHANGES
..
```

After every command in this format, EDITOR reports how many changes were made.

To correct the same error in more than one place, specify the range of lines in which the error exists:

```
../JR/=/UR/,120,140
  2 CHANGES
..
```

To change the position in which a character is located, you must use spaces. The tab character, which automatically positions to the tab stop in CREATE mode, has no special meaning in a text replacement string.

What happens if we try to use the tab character?

```
../0/=/;0/,160
  2 CHANGES
..LIST,160
 160=          ;03   FILLER PICTUPE X(1;05) VALUE SPACES.
..
```

We did not get the desired result, since the tab used in CREATE mode positions the line, but it never appears in a LIST. Now, to recover from the wrong correction:

```
../1;/=/1/,160
  1 CHANGES
../;0/=/     0/,160
  1 CHANGES
..LIST,160
 160=              03   FILLER PICTURE X(105) VALUE SPACES.
..
```

The blank insertion works the other way, also, to close up spaces.

```
••/     F/=/  F/,140
   1 CHANGES
••LIST,140
   140=        03  FILLER PICTURF X(105) VALUE SPACES.
••
```

If you are not sure of the number of the incorrect line, or if EDITOR could find it faster than you could, use the parameter ALL as a line number. EDITOR will search the entire edit file and find and change all occurrences of /oldtext/.

```
••/JR/=/UR/,ALL
   2 CHANGES
••
```

Figure 4-12 summarizes the changes made to the example. We have missed one error. The FILLER misspelling, but we'll catch that below.

```
••LIST,ALL
    100=        WORKING-STORAGE SECTION.
    110=        01  HEAD.
    120=            03  FILER PICTURE 9 VALUE 1.
    130=            03  FILLER PICTURE X(9) VALUE "JOB CLASS".
    140=            03  FILLER PICTURE X(17) VALUE SPACES.
    150=            03  FILLER PICTURE X(4) VALUE "NAME".
    160=            03  FILLER PICTURE X(105) VALUE SPACES.
••
```

Figure 4-12. Correction of Figure 4-11

COMMON PITFALLS IN USING /oldtext/=/newtext/

One skill to acquire in using the text replacement command, is selecting the proper /oldtext/ string. The idea is to find an /oldtext/ string that:

Uniquely identifies the string to be changed, yet

Reduces the number of characters to be entered in the command.

As the correction of the PICTURE misspelling showed, one way is to restrict the change to a range of lines. The required range parameter can be identified by any of the following:

Parameter	Resulting Range
line number	only line specified
line-start, line-end	first line specified through last line specified
LAST	only last line in file
line-start, LAST	first line specified through last line of file
ALL	entire file

Even with a line restriction, however, the choice of /oldtext/ characters must identify the precise string to be changed. Specifying too few characters in an attempt to be concise can lead to unexpected results.

For instance, the original lines 120 and 140 in figure 4-11 could have been changed:

```
..<u>/J/=/U/,120,140</u>
..  3 CHANGES
```

Logically, U should have replaced J only in the two instances where PICTURE was misspelled PICTJRE. But now line 130 would have been changed also; JOB CLASS would read UOB CLASS.

EDITOR makes the change requested, even if it is not what you wanted. Use the number-of-changes message to follow changes: if the number is not what you expected, LIST the file or line and recorrect as necessary.

Use the LIST command to verify the modified line is correct.

Two other optional parameters can place restrictions on the /oldtext/ to be changed. Both of these are available also on the DELETE, LIST, and SAVE commands.

A column range can be specified to restrict /oldtext/ to a portion of a line. Either a single column or a range of columns can be stated, with different interpretations existing for the range. These parameters have the format:

(column)

(column-1, column-2)

The enclosing parentheses distinguish columns from line range parameters.

The relation of the column range and the length of the /oldtext/ is:

If a single column is specified, the /oldtext/ character string must begin in that column.

If a range of columns is specified, the entire /oldtext/ character string must lie completely within that range.

As illustrations, consider lines from figure 4-11 and various means to change line 120 from FILER to FILLER:

/FILER/=/FILLER/,ALL,(16)	Change, since FILER begins in column 16.
/FIL/=/FILL/,ALL,(13-16)	No change, FILER begins, but does not end, within columns 13-16.
/FIL/=/FILL/,120,(8-20)	Change, since FILER lies within columns 8-20 in line 120.
/E/=/LE/,120	Produces desired FILLER, but also PICTURLE and VALULE.
/E/=/LE/,120,(19)	Corrects FILLER.

Though the (column) parameter is precise, it is not always practical to count to the column to be used in the command. The UNIT option offers some help.

UNIT	Specifies that /oldtext/ must not be preceded or followed by a letter or a digit.

UNIT rejects any /oldtext/ string that may be part of a word or variable name composed of letters or numbers. For example, consider the source program, complete with COBOL line numbers, that contains the following line, among others:

```
0350         03  DATA-NAME-1.
```

If you choose to restructure your program, changing all level 03 entries to level 05, you could enter the following:

```
/03/=/05/,ALL,UNIT
```

Only level numbers are changed, since the oldtext must be delimited by symbols, blanks, start-of-a-line, or end-of-line but not letters or numbers; line numbers will not change.

EDIT COMMAND FUNDAMENTALS

Any local file containing character data can be transferred to the edit file for updating. Since only one edit file can exist, execution of the EDIT command destroys the present edit file contents. EDITOR issues a warning message and forces you to re-enter the EDIT command if the edit file has not already been saved.

To transfer a local file to the edit file, use the EDIT command:

```
. . EDIT,file,SEQUENCE
```

Use of the optional SEQUENCE parameter depends on whether the local file has sequence numbers that EDITOR can use as edit file line numbers. SEQUENCE causes new line numbers to be assigned, beginning with 100 for the first line number and incrementing by 10 for each successive line number.

Text in the edit file must have line numbers. A file constructed through CREATE can be made a local file with its original line numbers or made local without line numbers, depending on SAVE command parameters:

. . SAVE,afile	Line numbers written to local file
. . SAVE,bfile,NOSEQ	Line numbers not written to local file

When file afile is returned to the edit file, line numbers already exist for EDITOR use and this EDIT command should be used:

```
. . EDIT,afile
```

In contrast, returning bfile to the edit file area requires:

```
. . EDIT,bfile,SEQUENCE
```

Figure 4-13 shows EDIT command use when another file currently exists in the edit file.

```
..FILES
--LOCAL FILES--
   WARBLER
..EDIT,WARBLER,SEQ
WARNING- EDIT FILE NOT SAVED
..SAVE,ORIOLE,NOSEQ
..EDIT,WARBLER,SEQ
..FILES
--LOCAL FILES--
   WARBLER      ORIOLE
..
```

Figure 4-13. Moving a Local File to the Edit File

Notice, a file moved to the edit file still remains as a local file.

linenum=text COMMAND FUNDAMENTALS

A single line in the edit file can be rewritten or added by identifying a line number. Format of this command is simply:

. . linenum=new line text

This command is an exception to the statement that EDITOR always prompts another response. At the conclusion of command execution, the carriage returns to the far left, but does not output two dots. You must enter the next command without prompting.

On entering the new line of text, the maximum line length, tab character, and tab stops defined by the FORMAT statement can be used. In figure 4-14, FORMAT,COBOL is assumed.

```
..236=;;DATA RECORD IS CARD
237=;;LABEL RECORD OMITTED.
LIST,ALL
   236=             DATA RECORD IS CARD
   237=             LABEL RECORD OMITTED.
..
```

Figure 4-14. Using line=text Command

Notice, both line 237 and the LIST command were entered with the carriage at the far left, since no prompting dots appear.

Lines can be entered in any order. EDITOR makes no distinction between adding a new line and rewriting an old line. If a single line is to be added but a free line number does not exist, renumber the lines in the existing file with the RESEQ command.

ADD COMMAND FUNDAMENTALS

There are times when you will want to add information to an existing program. Figure 4-15 illustrates a file that contains the basic elements of a COBOL program.

```
..CREATE,2,2
    2=;IDENTIFICATION DIVISION.
    4=;ENVIRONMENT DIVISON.
    6=;DATA DIVISION.
    8=;PROCEDURE DIVISION.
   10==
..
```

Figure 4-15. Program FRAME

Closer examination of this program shows that the file does not contain all of the basic constant entries in a COBOL program. For example, the STOP RUN statement is not included nor are the CONFIGURATION SECTION entries.

We want to add several lines between existing lines 4 and 6. Since any new line added to the file must have a unique line number and line numbers must be in ascending order, the file must be renumbered so that more than one line can be added after line 4. As shown in figure 4-16, as many as 14 lines can be inserted in the resequenced file.

```
..RESEQ,1,15
..LIST,ALL
     1=        IDENTIFICATION DIVISION.
    15=        ENVIRONMENT DIVISION.
    30=        DATA DIVISON.
    45=        PROCEDURE DIVISION.
..
```

Figure 4-16. Resequencing the Edit File

No EDITOR command exists to transfer a local file to the edit file while renumbering with specific line numbers, so resequencing is often necessary when substantial inserts are made.

When ADD is called, EDITOR presumes several lines are to be inserted as a group and displays the next available line number. (Use the linenum=text command if only a single line is to be inserted.) User entries at this point are similar to CREATE mode:

The current FORMAT command controls line length and tabs.

An equals sign must be entered to exit from ADD mode.

If lines were to be added at the end of the current file, command format would be:

```
. . ADD
```

If the insertion is to be made in the midst of the current file, for example, entries after the ENVIRONMENT DIVISION, the number of the first new line must be stated. The increment value for succeeding lines is optional, but normally should be used, since ADD does not overwrite or skip over existing lines. The increment value need not be the same as that used to create the file.

```
..ADD,16,1
  16=
```

Prompting continues after each user line entry, until one of the following occurs:

User enters = as first and only character in line.

The next line number equals or exceeds the value of a line currently in the file.

ADD has two other options discussed in section 6:

The SUP parameter inhibits prompting by line number display.

The OVERWRITE parameter allows existing lines to be overwritten.

Figure 4-17 shows insertion of new lines into an existing file.

```
..ADD,16,1
    16=;CONFIGURATION SECTION.
    17=;SOURCE-COMPUTER. CYBER.
    18=;OBJECT-COMPUTER. CYBER.
    19==
..
```

Figure 4-17. Adding Lines to Edit File

DELETE COMMAND FUNDAMENTALS

Any line existing in the current edit file can be eliminated with a DELETE command. Minimum parameters for the file specify the line or range of lines to be deleted.

```
. . DELETE,range
```

The required range parameter, as well as most of the optional parameters, are the same as for the LIST and SAVE commands. Options permit deletion of:

A single line of the edit file.

A range of lines.

All lines containing a particular character string.

All lines or all lines in a range containing a particular character string.

All lines or all lines in a range containing a particular character string in a particular position in each line.

In figure 4-17, we added lines to FRAME. Now, let's take them out. The easy way, when specific line numbers are known, is to identify the line to be deleted:

```
..DELETE,16,18
..
```

When you do not know the exact line number, but do know the specific arrangement of characters to be deleted, use DELETE with a search criteria.

```
..DELETE,ALL,/OBJECT/,(8)
1 DELETION
..
```

Any line in the edit file (lines to be searched specified by ALL) with the word OBJECT beginning in column 8 are deleted.

SUMMARY OF FILE UPDATING COMMANDS

If necessary, transfer local file to edit file; specify sequencing if the file was saved without line numbers.

```
..EDIT,BADFILE,SEQ
```

Display the file to determine specific changes to be made:

```
..LIST,300,LAST
    300= THIS LINE IS CORRECT
    310= MISTEAKS START
    320= MISTAKES START
    330= ERROR ARE IN HEER
    340= ANOTHER GOOOF
    350= THIS LINNE IS BAD TWO
    360= I CANT TYPE SO GOOD
    370= FINALLY OK
..
```

Begin file correction, choosing the command that best suits the corrections. Use the linenum=text command to overwrite an existing line or to insert an omitted line.

```
..330= ERROR IS IN HERE
```

Use DELETE to remove any extraneous lines, specifying a particular line number or a search criteria.

```
..DELETE,310
```

Use ADD to correct several lines in a row. The OVERWRITE parameter must be specified if existing lines are to be overwritten. ADD line incrementing continues until = is entered alone.

```
••ADD,340,10,OVERWRITE
    340= ANOTHER GOOF FIXED
    350= THIS LINE IS NOW OK
    360= I CAN TYPE BETTER
••  370==
```

Before adding lines, resequence if necessary.

```
••RESEQ
```

The updated file now reads:

```
••LIST,ALL,SUP
  THIS LINE IS CORRECT
  MISTAKES START
  ERROR IS IN HERE
  ANOTHER GOOF FIXED
  THIS LINE IS NOW OK
  I CAN TYPE BETTER
  FINALLY OK
••
```

PROGRAM INPUT AND OUTPUT FILES 5

The COBOL programs illustrated in section 4 have limited usefulness since each is coded for one specific action. Figure 4-3, for instance, prints three lines, but cannot be used to print another message without first changing the program and recompiling it. This limitation is why programs are usually written in such a way that data is independent of the program that uses it.

With INTERCOM, you have three options for providing a program with data:

1. Compile data into the program as in figure 4-3.

2. Write data onto a file and have the program read the file during execution.

3. Enter data from keyboard as program executes.

Option 1, as stated above, is valid for printing one particular set of data only, and is not practical in the long run.

Option 2, reading data from a file, requires that you first create the data and make it known to your program as a local file. Through the SELECT and FD statements, the file is identified in the program. Any file name, except a COBOL reserved word, can be used as the name of the data file. The COBOL reserved words INPUT and OUTPUT may be used as file names as long as they are used in accordance with the rules of COBOL; that is, the standard input and output files.

During execution, the data is read and manipulated according to Procedure Division statements. Reading data from a file stored in the system is fast and efficient. Moreover, since the data exists on the file in storage, the same data file can be used more than once or stored for future use. Independently of the program, the file can be displayed on a printing device to make a written record of the data.

If the data file was created through the EDITOR CREATE command, add the letters FZ to the file name in the SELECT clause to identify the file as one containing fixed length Z-type records, as follows:

```
SELECT AFILE ASSIGN AFILE-FZ
```

More detailed information concerning Z-type records can be found in the COBOL 4 reference manual.

Option 3, reading data input from the keyboard, involves the use of a connected local file. As with reading a local file, the program must establish the source and format of the expected data. COBOL offers two methods of handling connected files. The first method is simply to connect the input file and enter data from the keyboard. In this situation, the COBOL program looks exactly like one written for reading a local file. Each time a READ statement is executed, the program waits for data to be entered from the keyboard. The AT END imperative is executed if the program is coded to terminate on a specific input or %EOF is entered.

A second method to enter and output data is to construct the program using the ACCEPT and DISPLAY statements. In this situation, the source and format of the input data is defined in the Working-Storage Section. The use of ACCEPT and DISPLAY allows a program to be written in fewer statements, as shown later in this section. Less flexibility is achieved, however, since a program written in this manner cannot be used for batch processing without first modifying the SPECIAL-NAMES paragraph. When the DISPLAY verb is used, all output is single spaced. The first character is not treated as a carriage control character, as is normally done with output destined for the printer. Should a carriage control character be required, –C must be suffixed to the implementor name.

```
    SPECIAL–NAMES.
        TERMINAL–C IS HERE.
```

Reading data from the keyboard requires more overall execution time. The accuracy of keyboard entries is significant, since no error correction is possible once the RETURN key has been pressed to terminate the entry. No permanent record can be made of the data entered unless the program itself preserves the data in an output file. A second execution would require the data to be re-entered.

Options 2 and 3, executing with local file and connected file data, are described in the remainder of this section. EDITOR use is illustrated, but many of the same principles apply to advanced INTERCOM use when programs are executed without EDITOR.

The handling of files generated during execution is not discussed as a separate topic. Output file handling, whether the files are connected or disconnected, is generally analogous to the related input file handling.

REVIEW OF COBOL FILE LINKAGE

The portions of a COBOL program that link the program to external files are summarized here. COBOL treatment of external files is the same irrespective of INTERCOM status as a local or connected file, with two exceptions, the AT END imperative, as described above, and the fact that connected files cannot be rewound. The following description does not apply to interactive use of the ACCEPT and DISPLAY verbs.

External files are selected in the Environment Division, described in the Data Division, and read and/or written in the Procedure Division.

ENVIRONMENT DIVISION

The SELECT statement of the Environment Division, Input-Output Section links the COBOL program to the external file. In addition, this statement equates the COBOL program file name to the system file name. These names may be identical but need not be. COBOL file names are alphanumeric, may contain hyphens, and may be up to 30 characters long. System file names are limited to 7 characters and cannot contain special symbols, such as hyphens. Both names must begin with a letter and must not contain embedded blanks. The format of the SELECT statement is:

```
    SELECT filename1 ASSIGN TO filename2.
```

filename2 can be further described by appending a record description code. This code describes the type of records the file contains. All of these codes are described in the COBOL 4 reference manual; however, the code that you will probably use the most is FZ. This code describes fixed-length, zero-byte terminated records.

Any file that you create through the EDITOR CREATE command is created in this format. Failure to specify this type of record when reading files created through EDITOR causes your program to misread your data file. Should zero-byte terminators (::) appear in your output, check to see that you have properly defined your SELECT statement.

DATA DIVISION

All files selected in the Environment Division must be described in the Data Division File Section. The FD provides such information as the block size, file labels and the record names. (For files created through EDITOR, only the LABEL RECORDS OMITTED clause is necessary.) (Files to be sorted are similarly described by an SD paragraph.) Each FD or SD must be immediately followed by an 01 level description of the record. If required, this description may be further divided into subordinate level descriptions so that each field of the record may be described fully. The description must include the size of the fields and the type of data contained in the field; that is, numeric, alpha, or alphanumeric. File names must be unique; record and field names need not be. Record and field names must be planned carefully to prevent unnecessary coding of the Procedure Division. Non-unique names must be qualified and only duplicate names may be manipulated through the CORRESPONDING option.

PROCEDURE DIVISION

External files are read, written, and otherwise manipulated through Procedure Division statements. Prior to any program action, each file must be opened for input, output, or input-output. Files opened for input may be read but not written. Files such as those input from the card reader are always opened only for input. Files opened for output may only be written. Files such as those destined for the line printer are opened only for output. Non-sequential files stored on disks may be opened in any of the three ways. OPEN I-O can be used for existing files only.

All files opened by the program must be closed before program execution is terminated. In general, a file should be closed as soon as it is no longer required as this releases central memory space. Unless otherwise specified, COBOL automatically rewinds all files when they are opened and again when they are closed. Specifying a CLOSE file name WITH NO REWIND leaves the file positioned at the end of the last operation. A subsequent OPEN WITH NO REWIND statement followed by a WRITE statement causes a second set of data to be written on the file. The two data sets will be separated by an EOF. Since COBOL automatically attaches and opens files through its internal statements, it is not necessary to use INTERCOM commands to accomplish these functions.

EXECUTION WITH LOCAL DATA FILES

Thus far in this manual, file creation has implied entry of a COBOL program. It is not true, however, that the file entered through EDITOR must be a program. Any type of character data may be input. One of the more obvious uses of EDITOR, in addition to creating a program, is to create a file of data to be read by that program.

Although any local file can be connected, this discussion deals only with unconnected files. Names INPUT and OUTPUT are deliberately avoided, although these imply connected files when the EDITOR RUN command is used to execute a COBOL program.

Additional uses and parameters of the following commands are illustrated below.

FORMAT   RUN   DELETE

REWIND, CREATE, LIST, and SAVE commands are also used.

The first step in executing with a file of data is to create that file.

FORMAT CONTROL

The FORMAT command introduced in the previous section dealt only with the line length and tabs appropriate to a compiler language program. FORMAT is not limited to these values, however. The user can specify any line length, tab character, or tab stops.

FORMAT has only these restrictions:

Maximum line length is 510 characters.

The use of % as the tab character is not recommended. % is just as valid as any other tab character, but if the user enters %A or %S as data, the system will interpret these characters as a request for an abort or suspend.

Maximum number of tab stops is set by the individual installation, but is usually 10 stops.

FORMAT must be entered before CREATE is called, since interpretation of input characters during CREATE depends on the FORMAT in effect. Any user setting of FORMAT remains valid until another FORMAT is entered or until you exit from EDITOR.

To determine the current format specification, enter:

```
..FORMAT,SHOW
```

System response, assuming COBOL format, would be:

```
CH=72   TAB CHAR=; TAB COL= 8 12 16 20 24
    ↑              ↑        \____________/
 maximum      tab character    tab stops
 line length
```

You have the option to change any or all of these settings.

The command is:

```
. . FORMAT,CH=nnn,TAB=x,nn,nn, . . . nn
```

nnn	Maximum number of characters in each line
x	Character to act as tab
nn	Tab stop positions

For example, to establish a file to be read according to this FD:

```
01   INREC.
     05  FIELD1 PIC X(5).
     05  FILLER PIC X(8).
     05  FIELD2 PIC X(5).
```

Enter a maximum line length of 18 characters, accept the semicolon as a tab character, and set tab stop at 14 as shown in figure 5-1.

```
..FORMAT,CH=18,14
..FORMAT,SHOW

    CH=18       TAB CHAR=;       TAB COL= 14
..
```

Figure 5-1. Setting FORMAT for Data File

One other item, the relation between FORMAT and SAVE, is significant. The SAVE command has an optional parameter NOSEQ which strips line numbers before writing edit file contents to a local file. Since your COBOL program will probably be reading numeric values, use the NOSEQ parameter when making your EDITOR created data file a local file so that the line numbers cannot be inadvertently read as part of the data.

Consider a line with text HERE ARE 16 LTRS. When this line is saved with sequence numbers, and the above format, (NOSEQ parameter is omitted) it exists as:

```
HERE ARE 16 LTRS   000100
```

Saved with the NOSEQ parameter, it is simply:

```
HERE ARE 16 LTRS
```

Figure 5-2 shows creation of a file with four sets of data to be read in the format specified by INREC:

```
..FORMAT,CH=18,14
..CREATE,SUP
ENTER LINES

12345;12345
22222;22223
00033;00033
66 66;66666
=
..SAVE,AFILE,NOSEQ
..
```

Figure 5-2. Data File Creation

EXECUTION THROUGH RUN COMMAND

Now that a file of data exists and has been saved as a local file with the name AFILE, we must create the COBOL program that will read the data. It is shown in figure 5-3. Enter the program and make any input corrections as explained in section 4.

```
         ID DIVISION.
         PROGRAM-ID. DATARD.
         ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
         SOURCE-COMPUTER. CYBER.
         OBJECT-COMPUTER. CYBER.
         INPUT-OUTPUT SECTION.
         FILE-CONTROL.
         SELECT AFILE ASSIGN AFILE-FZ.
         SELECT OUTFILE ASSIGN OUTFILE-FZ.
         DATA DIVISION.
         FILE SECTION.
         FD  AFILE
             LABEL RECORD OMITTED DATA RECORD IS INREC.
         01  INREC.
             05  FIELD1 PIC X(5).
             05  FILLER PIC X(8).
             05  FIELD2 PIC X(5).
         FD  OUTFILE
             LABEL RECORD OMITTED
             DATA RECORD IS OUTREC.
         01  OUTREC PIC X(18).
         PROCEDURE DIVISION.
         START.
             OPEN INPUT AFILE.
             OPEN OUTPUT OUTFILE.
         LOOP.
             READ AFILE AT END GO TO WRAP-UP.
             IF FIELD1 IS EQUAL TO FIELD2 MOVE INREC TO OUTREC
             WRITE OUTREC.
             GO TO LOOP.
         WRAP-UP.
             CLOSE AFILE OUTFILE.
              STOP RUN.
..RUN,COBOL
   COMPILING DATARD
   000 E AND      T/U DIAGNOSTICS ISSUED
        053377B SCM USED
             .315 CP SECONDS COMPILATION TIME
 END COBOL
..
```

Figure 5-3. Program DATARD to Read Local File

Given that no compilation errors or execution errors were reported and no output other than that shown above is displayed at the terminal, it appears that execution was successful. But where is the output the program wrote?

Executing the FILES command before the RUN command would have shown only one local file. AFILE, which contains the data to be read in the program. Executing FILES after the RUN command shows five files:

```
..FILES
--LOCAL FILES--
AFILE      $INPUT      $OUTPUT      LGO      OUTFILE
..
```

INPUT and OUTPUT are the connected files always produced by RUN execution even if, as in this example, they are not used by the program. File named LGO contains the compiled program. OUTFILE should contain the information output by the program.

How can we read OUTFILE? The easiest way is to move OUTFILE to the edit file and list it. Remember that EDITOR does not let you destroy the edit file inadvertently, so if you do not save the current edit file before moving OUTFILE, you receive a warning message and must re-enter the EDIT command.

Figure 5-4 shows how the local file OUTFILE can be listed

```
..SAVE,DATARD
..EDIT,OUTFILE,SEQUENCE
..LIST,ALL,SUP
 12345          12345
 00033          00033
..
```

Figure 5-4. Listing a Local Output File

The SUP parameter of LIST suppresses display of the EDITOR line numbers assigned with EDIT. Without the SUP parameter, the first line would appear:

```
..LIST,100
  100=12345          12345
..
```

The numbers are necessary for putting a file in the edit file. The fact that an edit file is designed for updating does not mean that its features cannot be used for a simple listing of the file contents.

SECOND EXECUTION

When only the files INPUT and OUTPUT are used for a program executed through the RUN command, you need not reposition files. EDITOR rewinds the files OUTPUT and LGO each time RUN is called. Successive executions of a program require no special actions.

Similarly, COBOL rewinds files, except connected files, whenever an OPEN or CLOSE statement is executed, unless you have written statements to prevent the rewind. Consequently, you need not concern yourself with file positioning as long as you are using the RUN command and COBOL. A second execution of the program can be accomplished without recompiling. Remember that the RUN execution created a file LGO that contains the compiled program. You can execute LGO by entering its name as a command:

```
••LGO
```

Since LGO is a local file containing a binary program (not a coded source program), it executes the same statements as those executed from the RUN command that created it. Again, since input and output file positioning is automatically performed by COBOL, and LGO is rewound by the operating system, all required files have been rewound during execution and no diagnostics are issued.

If the program had specified the following:

```
OPEN INPUT AFILE WITH NO REWIND
OPEN OUTPUT  OUTFILE WITH NO REWIND
•
•
CLOSE AFILE OUTFILE WITH NO REWIND
```

the second execution would not have been successful. The reason is:

At the end of execution through the RUN command, files AFILE and OUTFILE were positioned at the end of the last set of data read or written, the automatic rewind having been disabled by the WITH NO REWIND option. The LGO command caused the program to attempt to continue reading from AFILE, but instead of data, the program encountered an end-of-information terminator and the AT END imperative was executed.

To execute a program a second time, the data file must be positioned to accommodate that second execution. If the same set of data is to be used for execution, rewind the file before execution begins. If another set of data is to be executed, the original input file must be established with more than one set of data, as illustrated in the next topic.

Repositioning a file at its beginning can be accomplished by an INTERCOM command:

```
. . REWIND,file
```

More than one file can be named, as in:

```
REWIND,file,file,file, . . .
```

Within the program, file positioning is limited to rewinding or not rewinding files. A file closed with no rewind may be subsequently opened either in its current position (by specifying OPEN. . .WITH NO REWIND) or rewound, by omitting the no rewind option.

Figure 5-5 shows two executions of the same set of data. PROGRAM DATARD is assumed to exist as a local file saved in figure 5-4, modified to include the NO REWIND options as shown above.

```
••RUN,COBOL,FILE=DATARD
 •
 •
••REWIND,AFILE
••LGO
••
```

Figure 5-5. Rewind Between Executions of Same Program

Notice, the file written by the program, OUTFILE, was not rewound. With the NO REWIND clause specified, rewind of output file with an INTERCOM command is optional, depending on the results desired.

Let's examine OUTFILE, as we did in figure 5-4, by transferring it to the edit file and listing. We find two copies of the program results.

```
••EDIT,OUTFILE,SEQUENCE
••LIST,ALL
     100=12345              12345
     110=00033              00033
     120=*EOF
     130=12345              12345
     140=00033              00033
••
```

Any data written during program execution always is written at the current file position. In the absence of the automatic rewind, the current position is after the last data written. At the end of a program, the output file is closed and a terminator written after current data. No repositioning occurs, however, and any successive write to a file with the same name continues after the terminator. Although not shown above, the terminator written at file close becomes the characters *EOF in the edit file.

If a file is rewound before the second set of data is written, current position is beginning of information. Any existing data, in this instance, is destroyed by new data written over the first set of data.

Rewind an output file to write over existing data; do not rewind if existing data is to be preserved and new data is to be written at the end. Remember, the preceding example applies only to programs that contain both OPEN and CLOSE WITH NO REWIND statements.

EXECUTION WITH TWO SETS OF DATA ON ONE FILE

All local data files created through EDITOR have special system symbols at the end to mark the end of the file. These markers are supplied by EDITOR without user action. When such a file is read by an executing program, the file ending marker can be sensed by the AT END imperative. Through the use of an *EOF in EDITOR, you can write a similar marker into a file under construction with CREATE, effectively, dividing the file into two parts.

Figure 5-6 shows a data file being constructed for a program shown in figure 5-7.

```
••CREATE
   100=000123
   110=000123
   120=000123
   130=000069-
   140=      S
   150=000400
   160=000200
   170=      S
   180=001000
   190=*EOF
   200=111111
   210=111111
   220=111111
   230=111111
   240=*EOF
   250=125400-
   260=000258-
   270=000147
   280=*EOF
   290=000100
   300=      U
   310=*EOF
   320=   123
   330=   456
   340=      T
   350==
••SAVE,INFILE,NOSEQ
••
```

Figure 5-6. *EOF Use in Data for Program TENKEYDISK

Program TENKEYDISK is designed to read and total multiple sets of data from one file. There is no limit to the number of sets that can be processed since processing control is retained by the terminal user. The data sets, as created in figure 5-6, are separated by an *EOF†. This separator is detected by the AT END imperative and control is passed to the CLOSE-PARA which interrogates the terminal to determine if the next set is to be processed. Program termination does not occur until the terminal user answers NO (or, in fact, answers with anything but YES). Notice that the program, as coded, goes to the CLOSE-PARA when either a T (total) request is encountered or the AT END imperative is activated by the *EOF. Consequently, using the T prior to an *EOF causes the total to be printed twice, the first time with the correct number and the second with a contents of 0. The last data set in the example contains this condition and, as you will notice in figure 5-8, an extra total line is printed. This proves that the *EOF condition is recognized by the COBOL AT END imperative.

†When the file is saved on mass storage, an end-of-record level 17 is written in the *EOF position.

```
ID DIVISION.
PROGRAM-ID. TENKEYDISK.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 6600.
OBJECT-COMPUTER. 6600.
SPECIAL-NAMES,
    TERMINAL IS HERE.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INFILE ASSIGN INFILE-FZ.
DATA DIVISION.
FILE SECTION.
FD  INFILE
    LABEL RECORD OMITTED
    DATA RECORD IS AMOUNT.
01  AMOUNT.
    02  AMT PIC 9(6).
    02  SYNE PIC X.
WORKING-STORAGE SECTION.
01  ACCUM PIC S9(6).
01  ANS PIC X.
01  REG PIC ------9.
PROCEDURE DIVISION.
FIRST-PARA.
    OPEN INPUT INFILE.
    MOVE ZEROS TO ACCUM REG.
SEC-PARA.
    READ INFILE RECORD AT END GO TO CLOSE-PARA.
    IF SYNE = "T" GO TO CLOSE-PARA.
    IF SYNE = "S" GO TO SUBTOTAL-PARA.
    IF SYNE = "-" GO TO SUBTRACT-PARA.
    IF AMT NOT NUMERIC
        DISPLAY "BAD DATA" UPON HERE GO TO SEC-PARA.
    ADD AMT TO ACCUM
      ON SIZE ERROR DISPLAY "ACCUMULATOR OVERFLOW" UPON HERE
        GO TO SEC-PARA.
SUBTRACT-PARA.
    SUBTRACT AMT FROM ACCUM
      ON SIZE ERROR DISPLAY "ACCUMULATOR OVERFLOW" UPON HERE
        GO TO SEC-PARA.
SUBTOTAL-PARA.
    MOVE ACCUM TO REG.
    DISPLAY "SUBTOTAL IS  " REG UPON HERE.
    GO TO SEC-PARA.
CLOSE-PARA.
    MOVE ACCUM TO REG.
    DISPLAY "TOTAL IS  " REG UPON HERE.
    MOVE ZEROS TO ACCUM.
    DISPLAY "MORE" UPON HERE.
    ACCEPT ANS FROM HERE.
    IF ANS = "Y" MOVE ZEROS TO ACCUM
      CLOSE INFILE WITH NO REWIND
        OPEN INFILE WITH NO REWIND
          GO TO SEC-PARA.
    CLOSE INFILE.
    STOP RUN.
```

Figure 5-7. Program TENKEYDISK to READ Multiple Sets of Data

You will also notice that the last data set contains the numbers 123 and 456 rather than 000123 and 000456. A space and a zero are not recognized as equivalent by the COBOL compiler. Since the space is a non-numeric character, the IF AMT NOT NUMERIC sentence is activated and the words BAD DATA are displayed. This prevents spurious data from being added to the accumulator. Tests of this type are considered to be good coding practice in COBOL.

Notice also that the FILE-CONTROL SELECT statement assigns the COBOL file INFILE to the operating system file INFILE-FZ. The -FZ suffix informs the compiler that INFILE contains zero-byte terminated records, as are all EDITOR created records.

Each time the terminal user elects to process the next set of data, INFILE is closed and immediately reopened without being rewound. Although this action does not change the file positioning, it is necessary because a file cannot be read after an AT END condition is detected without the intervening CLOSE and OPEN.

```
••RUN,COBOL
 COMPILING TENKEYD
  000E AND      T/U DIAGNOSTICS ISSUED
      053377B SCM USED
       .308 CP SECONDS COMPILATION TIME
 END COBOL
SUBTOTAL IS        300
SUBTOTAL IS        900
TOTAL IS          1900
MORE
?      Y
TOTAL IS        444444
MORE
?      Y
TOTAL IS       -125511
MORE
?      Y
BAD DATA
TOTAL IS           100
MORE
?      Y
BAD DATA
BAD DATA
TOTAL IS             0
MORE
?      Y
TOTAL IS             0
MORE
?      N
••
```

Figure 5-8. Execution of Program TENKEYDISK

EXECUTION WITH CONNECTED FILES

A connected file is a local file with special characteristics. Data on the file exists only as characters passed between the terminal and an executing program. No copy of the data remains in the system.

In figure 5-3, a program named DATARD reads data from AFILE. How is execution affected by connecting AFILE1 before execution?

```
••CONNECT,AFILE
••FILES
--LOCAL FILES--
$AFILE       DATARD
••RUN,COBOL,FILE=DATARD
COMPILING DATARD
  000E AND      T/U DIAGNOSTICS ISSUED
     053377B SCM USED
          .335 CP SECONDS COMPILATION TIME
 END COBOL
```

After waiting a reasonable period of time, we are apt to decide that RUN execution is hung, but not so. The program is in execution and expecting keyboard entry of data. This fact can be shown by trying any of the ploys otherwise used to test whether the system is still responsive to keyboard input; normally, entering a RETURN key results in a line feed.

Program DATARD expected another line of data and is still waiting for input. Since the program is expecting data in the format X(5), 8 filler, X(5), almost anything you enter is accepted, and it appears that nothing has happened.

Consequently, when writing a program to be executed with connected input files, it is well to prompt response by including DISPLAY or WRITE statements that display, at minimum, the fact that input is expected. Even display of a single character such as a question mark should alert the terminal user to the current situation. Of course, if you are using the ACCEPT and DISPLAY verbs to process your I/O, rather than READ and WRITE, the question mark is displayed automatically.

Now rewrite program TENKEYDISK as a program named TENKEYI. The example in figure 5-9 performs the same operations as shown in figure 5-7.

To execute program TENKEYI, use the RUN command. Since all input and output is accomplished by the COBOL ACCEPT and DISPLAY commands, you need not connect or create any files. The keyboard becomes your input file and the display becomes your output file when you equate the terminal with the object of the ACCEPT and DISPLAY verbs in the SPECIAL–NAMES paragraph. In fact, the system is using the INPUT and OUTPUT files, which are automatically connected by RUN.

Each time the program expects input, a question mark is displayed. This action is automatic; it occurs each time the ACCEPT verb is encountered. However, there are times when it would be advantageous to display more prompting information. In this example, we have elected to display the words ENTER DATA at the beginning of program execution. The code could have been written so that the ENTER DATA was displayed each time input was expected. However, since the compiler is going to issue the question mark as a separate line, this would mean that two lines would be output for each line of expected input. Such action would slow execution.

```
ID DIVISION.
PROGRAM-ID.  TENKEYI.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 6600.
OBJECT-COMPUTER. 6600.
SPECIAL-NAMES. TERMINAL IS HERE.
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01   ACCUM PIC S9(7).
01   ANS PIC X.
01   REG PIC ------9.
01   AMOUNT.
     03   AMT   PIC 9(6).
     03   SYNE PIC X.
PROCEDURE DIVISION.
FIRST-PARA.
     DISPLAY "ENTER DATA" UPON HERE.
     MOVE ZEROS TO ACCUM REG.
SEC-PARA.
     ACCEPT AMOUNT FROM HERE.
     IF SYNE = "T" GO TO CLOSE-PARA.
     IF SYNE = "S" GO TO SUBTOTAL-PARA.
     IF SYNE = "-" GO TO SUBTRACT-PARA.
     IF AMT NOT NUMERIC
     DISPLAY "BAD DATA" UPON HERE GO TO SEC-PARA.
     ADD AMT TO ACCUM
     ON SIZE ERROR DISPLAY "ACCUMULATOR OVERFLOW" UPON HERE.
     GO TO SEC-PARA.
SUBTRACT-PARA.
     SUBTRACT AMT FROM ACCUM
     ON SIZE ERROR DISPLAY "ACCUMULATOR  OVERFLOW" UPON HERE.
     GO TO SEC-PARA.
SUBTOTAL-PARA.
     MOVE ACCUM TO REG.
     DISPLAY "SUBTOTAL IS  " REG UPON HERE.
     GO TO SEC-PARA.
CLOSE-PARA.
     MOVE ACCUM TO REG.
     DISPLAY "TOTAL IS  " REG UPON HERE.
     DISPLAY "MORE" UPON HERE.
     ACCEPT ANS FROM HERE.
     IF ANS = "Y" MOVE ZEROS TO ACCUM
     GO TO SEC-PARA.
     STOP RUN.
```

Figure 5-9. Use of ACCEPT and DISPLAY

If you prefer, similar results could be accomplished by using program TENKEYDISK and connecting the input file, INFILE. (The output already uses the terminal as a display medium.) Since all input is activated by the READ command, rather than ACCEPT, the question mark is not displayed. In fact, no indication is given that the program is in execution and awaiting input. Here, it would be wise to issue at least one prompting message such as ENTER DATA. Subsequent input may then be entered as fast as you can key since INTERCOM stacks the input in a buffer until your program is ready for it.

CONNECT AND DISCONT COMMAND FUNDAMENTALS

Any local file can be connected by the INTERCOM command CONNECT:

```
CONNECT,file
```

More than one file can be connected by naming several files in the command.

The command that disconnects a file is:

```
DISCONT,file
```

Again, more than one file can be named in the command. Disconnected files are not returned. Rather, the file continues to exist and any additional data written to the file is preserved on mass storage.

EXPERIMENTING WITH CONNECTED FILE INPUT

One of the ways to learn connected file operation is to enter data at different times and study system response, as in the following examples. Program TENKEYDISK (figure 5-7) is presumed to be in the edit file.

WHAT HAPPENS IF INFILE HAS NOT BEEN CONNECTED?

```
..RUN,COBOL
 COMPILING TENKEYD
  000 E AND      T/U DIAGNOSTICS ISSUED
     053377B SCM USED
        .375 CP SECONDS COMPILATION TIME
  END COBOL
  INFILE - OPEN OLD ON A NON-EXISTING FILE
    SOURCE LINE NUMBER 00024
    FATAL COBOL ERROR - RUN TERMINATED
  ..
```

Execution of the RUN command causes files named INPUT and OUTPUT to be connected. Any other file to be entered through the keyboard must be connected by the user.

```
..CONNECT INFILF
```

WHAT HAPPENS IF ANOTHER COMMAND IS ENTERED BEFORE RUN COMPLETES?

```
••RUN,COBOL
••Q
 COMPILING TENKEYI
  000 E AND     T/U DIAGNOSTICS ISSUED
     053377B SCM USED
          •335 CP SECONDS COMPILATION TIME
 END COBOL
ENTER DATA
?
BAD DATA
•
•
```

Data entered must conform to the format expected by the program. In this case, all input must be numeric. Any INTERCOM or EDITOR command is accepted simply as if it were the expected data.

CAN INPUT BE ENTERED BEFORE THE PROGRAM PROMPTS INPUT?

Yes, input can be entered before any program output is displayed. You cannot inhibit display of any prompting message (including the question mark if ACCEPT is being used) by such action, however.

CAN TWO LINES OF INPUT BE ENTERED TOGETHER?

Any data input must conform to the format expected by the program.

```
••RUN,COBOL
 COMPILING TENKEYI
  000 E AND     T/U DIAGNOSTICS ISSUED
       053377B SCM USED
            •335 CP SECONDS COMPILATION TIME
 END COBOL
ENTER DATA
000752000003
000000T
TOTAL IS          752
MORE
?    YES
000752-000003-
       T
TOTAL IS         -752
MORE
?    NO
••
```

The program is expecting six numbers and/or an alphanumeric entry in the seventh character position. Consequently, anything after the seventh character is discarded.

ENDING CONNECTED FILE INPUT

When reading is from an unconnected file, normally the AT END imperative is included in the program to test for end of data (EOF). This same condition can be duplicated from interactive keyboard input by entering %EOF. %EOF, then, is the interactive equivalent of a 6/7/8/9 card. Similarly, %EOR may be entered to perform the function normally performed by the 7/8/9 card.

Another way to stop interactive READ processing is to test for a particular data input, and if that input is encountered, stop further read attempts. Remember that any end-of-data indicator you choose must conform to the format expected by your program. The example program interrogates the user each time a total is requested. If more data is to be processed, a Y (yes) must be input. If any other response is encountered, including NO, the program will terminate. This method provides the user with more flexiblity in handling data.

One more item to remember; if you neglect to program an end to your requests for input through the terminal, you can always stop execution through an abort by entering %A.

SUMMARY OF CONNECTED FILE OPERATIONS

Any input or output file may be used interactively by connecting that file prior to program execution.

```
••CONNECT,AFILE
••RUN,COBOL
```

Once connected, any data that was in the file is no longer available until the file is disconnected. Any data written to a connected file is lost once it is displayed. A file may be disconnected by the DISCONT command.

```
••DISCONT,AFILE
```

Avoid the use of INPUT and OUTPUT as file names since these files are automatically connected by the RUN command. If these file names are to be used, execute your program as follows:

```
••COBOL(I=filename)
••LGO
```

The use of the COBOL ACCEPT and DISPLAY verbs provides the following features:

1. The terminal is connected automatically through the COBOL provided file, TERMINL.

2. The program need not contain a File Section. Only a Working-Storage Section entry need be provided.

3. A prompting message (?) is automatically displayed each time the ACCEPT verb is executed.

INTERCOM AND EDITOR COMMANDS 6

This section presents INTERCOM and EDITOR commands in alphabetical order including the special EDITOR entries *EOR and *EOF. It tells when the command is used and gives the complete format. Errors often made in using the command, as well as recovery from these errors, are described.

All EDITOR commands are included.

INTERCOM commands included are:

EDITOR which makes file creation and editing possible

STORE, FETCH, and DISCARD for permanent file operations

FILES for status information

CONNECT and DISCONT for connecting and disconnecting files

BATCH to change status or file name

TEACH to describe INTERCOM operation

LOGIN and LOGOUT which establish INTERCOM access, are described in section 2.

Commands corresponding to operating system control statements are included only if they can be used by persons unfamiliar with their concepts. Commands included are:

AUDIT	to obtain permanent file status
RETURN	to eliminate files
REWIND	to rewind files

All options for the commands described are included. Examples are given only for parameters frequently used by inexperienced INTERCOM users. Diagnostic or error responses are included, since they point out command restrictions or common errors.

ADD COMMAND

ADD is an EDITOR command that allows lines to be added to an existing edit file. Although ADD can be used to add a single line in the midst of the file, use of another command, linenum=text, is preferable for inserting only one line.

The system response to ADD establishes an operation similar to CREATE:

Automatic incrementing of line numbers is in effect, with either the default or user specified incrementing value.

User input for the next line is prompted by display of the line number, unless prompting is suppressed by the SUP option.

An equals sign must be entered as the first and only character in the line to leave the ADD mode.

WHEN IS ADD USED?

To insert more than one line as a group within an existing edit file.

To make additions to the end of an existing edit file.

No new lines can be added unless unused line numbers exist or existing lines are to be destroyed. If necessary, use the RESEQ command to change existing line numbers before calling ADD.

ADD FORMAT

In the simplest form:

ADD	abbreviated A

EDITOR assumes that lines are to be added to the end of the existing file and prompts user input by displaying a line number with a value 10 greater than the last existing line. Prompting continues after each user entry until an equals sign is entered as the only character in a line.

```
ADD,SUP,OVERWRITE,start,increment
```

Optional parameters:

SUP	abbreviated S Suppresses prompting with line number display. The system response is ENTER LINES. Individual lines are to be entered in succession.
OVERWRITE	abbreviated O Existing line numbers are overwritten if necessary. Without this parameter, an insert into the file is not accepted if an existing line number is encountered.
start	Point at which line numbers should begin. Default is 10 greater than the number of the last line in the existing file.
increment	Number added to starting line to determine next and succeeding line numbers. Default is 10. This parameter cannot be used unless start is also specified.

The values for the start and incrementing of line numbers need not be the same as those used in the remainder of the file:

```
..CREATE
  100=THIS LINE WAS PART OF ORIGINAL FILE
  110=SINCE CREATE DID NOT SPECIFY LINE START OR INCREMENT
  120=DEFAULT LINE NUMBERS START AT 100 AND
  130=INCREMENT BY 10
  140==
..ADD,231,1
  231=LINES ADDED BY THE ADD COMMAND
  232=NEED NOT CONTINUE THE INCREMENT OF THE ORIGINAL FILE
  233=THE START NUMBER FOR ADD,HOWEVER, MUST BE GREATER
  234=THAN LAST NUMBER OF ORIGINAL FILE
  235=TO EXIT FROM ADD, ENTER AN EQUALS SIGN
  236=JUST AS CREATE IS ENDED
  237==
..
```

ADD does not skip over any existing line, nor does it allow any existing line to be rewritten, unless the OVERWRITE parameter is included in the ADD command. Either a skip or rewrite attempt terminates ADD.

```
..CREATE,550
  550=A START LINE NUMBER WAS SPECIFIED
  560=SINCE THE INCREMENT WAS NOT SPECIFIED
  570=LINE NUMBERS INCREMENT BY 10
  580==
..ADD,551
  551=     I AM TRYING TO ADD LINES BETWEEN 550 AND 560
ADD WONT REPLACE OR BYPASS LINES.
..
```

At first glance, you may assume that line 551 is not in the file; not so. The diagnostic tells you the next line to be entered – 561 – is not permitted since an existing line occurs between 551 and 561. Default incrementing for both the existing file and the ADD insert creates a problem. LIST shows 551 has been made part of the file.

```
..LIST,550,560
  550=A START LINE NUMBER WAS SPECIFIED
  551=     I AM TRYING TO ADD LINES BETWEEN 550 AND 560
  560=SINCE THE INCREMENT WAS NOT SPECIFIED
..
```

The proper way to add lines to the preceding file is to specify a start and increment value and exit from ADD with an equals sign:

```
..ADD,552,1
  552=BY CHANGING THE INCREMENT, LINES CAN
  553=BE ADDED BETWEEN EXISTING LINES
  554=AS LONG AS UNUSED LINE NUMBERS EXIST
  555==
..
```

On the other hand, you can take advantage of the automatic termination of ADD when you try to bypass an existing line as follows:

```
  ..ADD,561,2
     561=THESE ARE 5 NEW LINES
     563=ADDED BETWEEN 560 AND 570
     565=I WONT HAVE TO EXIT WITH =
     567=SINCE THE LAST LINE I WANT
     569=IS THE LAST LINE ADD WILL ACCEPT IN THIS RANGE
  ADD WONT REPLACE OR BYPASS LINES.
  ..
```

AUDIT COMMAND

AUDIT is an operating system command that gives statistics about a user's files made permanent by the STORE command or the CATALOG command.

WHEN IS AUDIT USED?

To find the name needed to access a permanent file.

To obtain status information such as when your file can be destroyed or copied to an archive tape.

AUDIT FORMAT

Parameters can appear in any order. AUDIT has no abbreviation.

AUDIT,AI=P,LF=lfn,ID=owner identification

P	Required letter.
lfn	Name of file to receive AUDIT output. 1-7 letters or digits beginning with a letter. Normally, the LF parameter is the name of a connected file. If the LF parameter is omitted, statistics are written to the file OUTPUT.
owner identification	Name used on STORE commands to make file permanent. 1-9 letters or digits.

The ID parameter is the same as that used on a STORE or FETCH command.

```
STORE,TEST,MINE              FETCH,TEST,MINE
           |_______________| |______________|
         AUDIT,LF=lfn,ID=MINE,AI=P
```

SYSTEM RESPONSE TO AUDIT

AUDIT execution writes status information to the file named with the LF parameter.

The following sample was produced on a Teletype that has a maximum of 72 characters on a line. AUDIT output is formatted for line printer output with 135 print positions. Consequently, terminal output requires 2 lines of text for every single line of printer output.

```
  AUDIT OF 6000 PERMANENT FILES  PARTIAL   ID              TIME
 10.48.38       10/09/75                 PAGE NO.      1

  SETNAME=PFQSET:

  OWNER          PERMANENT FILE NAME                   CYCLE  ACCOUNT
    VSN   PRUS   CREATION   EXPIRATION   LAST ATT    LAST ALT
   HAWK          NEWTEXT                                         999
 00844L         1  10/09/75   10/14/75     10/09/75    10/09/75
   HAWK          MASTER                                          999
 00844K         1  10/09/75   10/14/75     10/09/75    10/09/75


  AUDIT FINISHED
 EXIT
```

Read every other line of terminal output, giving a logical table with 2 entries for files NEWTEXT and MASTER:

OWNER	PERMANENT FILE NAME	CYCLE	ACCOUNT	UNIT	PRUS	CREATION
HAWK	NEWTEXT	999		844	1	10/09/75
HAWK	MASTER	999		844	1	10/09/75

The time and date of AUDIT execution appears in the output heading. Table entries show:

OWNER is your name which is specified by the ID parameter of the AUDIT command. Every file referenced on a STORE command with your name will appear.

PERMANENT FILE NAME is the name of a file referenced on a STORE command.

CYCLE is a system assigned number. Two permanent files with the same name are distinguished by cycle numbers. (These cannot be created through STORE, since cycle 999 is always created.)

ACCOUNT, UNIT, and PRUS refer to the hardware on which the file exists and are not significant at this time.

CREATION is the month, day, and year the file was made permanent.

EXPIRATION date shows the end of the file retention period.

LAST ATT is the date the file was last called for use.

LAST ALT is the date the file was last modified and may be the same as the creation date for files made permanent by STORE.

When AUDIT is complete, this line appears:

```
EXIT
```

OUTPUT FILE NAME

AUDIT output can be written to any file named with LF parameter. INTERCOM is used most effectively when an AUDIT command is preceded by a CONNECT command. Then, as AUDIT generates output, it is displayed directly at the terminal. Typical use is:

```
CONNECT,RESULTS

AUDIT,LF=RESULTS,AI=P,ID=ME
```

The file named RESULTS is displayed.

Omitting an LF parameter produces the same effect as stating LF=OUTPUT.

If the output file is not connected, AUDIT results are still written to the file named. In this instance, the only indication of AUDIT completion is the word EXIT.

```
COMMAND-AUDIT,ID=HAWK,AI=P,LF=PFIL
  EXIT
COMMAND-FILES
--LOCAL FILES--
  PFIL
```

To examine PFIL, you could list the file under EDITOR control or rewind and copy it to another connected file.†

```
COMMAND-CONNECT,LOOK
COMMAND-REWIND,PFIL
COMMAND-COPY,PFIL,LOOK
```

BATCH COMMAND

BATCH is an INTERCOM command that directs disposition of a file previously created or saved by the user. Except for the LOCAL disposition, the file must be a local file or an attached permanent file accessible to the terminal user. If an attached permanent file is specified with any disposition of the BATCH command except RENAME, a copy of the file is created and processed by BATCH; the attached permanent file remains unchanged. For other local files, no copy is made, and except for LOCAL and RENAME dispositions, the file does not exist as a local file following execution of BATCH.

†Since AUDIT results are formatted for a printer, the output can be referenced in a COPY utility command. (When a file has not been formatted as a print file, COPYSBF should be used in place of COPY.)

WHEN IS BATCH USED?

Beginning INTERCOM users find BATCH necessary for four situations:

To send a file from a terminal with no printing display to another terminal that has a printing display. A file created at a CRT terminal, for example, can be sent to a 200 User Terminal for output.

To change a REMOTE OUTPUT file received from another terminal to a LOCAL file for a printing display.

To change the name of the INPUT file so that it can be released from the terminal.

To change the name of any local file to some other name.

Experienced programmers familiar with the operating system and job deck structure use BATCH to submit a full job for execution at the central site. The TEACH utility and the INTERCOM reference manual provide instruction for remote batch operations.

BATCH FORMAT

BATCH is an interactive command: when called without parameters, it prompts specific responses until the user enters END to terminate BATCH use.

To initiate BATCH, enter this word and wait for system response. The word BATCH cannot be abbreviated.

```
BATCH
```

The system response message is:

```
TYPE FILE NAME–
```

Your response should be the name of a file.(Remember the RETURN key.)

The next system message is:

```
TYPE DISPOSITION–
```

Disposition should be LOCAL, RENAME, PUNCH, PUNCHB, or PRINT, depending on your reason for calling BATCH. A third message TYPE FILE ID is displayed for a PRINT, PUNCH, or PUNCHB disposition.

After one file has been changed in disposition, BATCH prepares to deal with another file and again displays:

```
TYPE FILE NAME –
```

At this point you can enter END and a RETURN key to terminate BATCH, or can continue with the name of another file to be processed with the same or different disposition.

As an alternate to BATCH operation with system prompting, you can enter all parameters at once, as in:

```
BATCH,file,LOCAL
```

PRINT DISPOSITION OF BATCH

The PRINT disposition places a file into the remote output queue and prints a hardcopy of the file at a terminal that is logged in. Use it when your terminal has a CRT screen but you have access to a 200 User Terminal.

After you enter BATCH,filename, INTERCOM prompts you as follows:

```
TYPE DISPOSITION -
```

Type the following.

```
PRINT,x
```

x	The 2-character terminal or user identifier of the terminal to receive the file, the word HERE to indicate the terminal identifier, or the word MINE to indicate the user identifier. The default is the central site.

You determine the terminal or user identifier of the receiving terminal by checking the messages that appear after logging in at the receiving terminal or by entering the ASSETS command at the receiving terminal. Both methods are shown as follows:

```
08/18/75   LOGGED IN AT 11.03.02.
           WITH USER-ID CR ----------- Use this identifier in command:
           EQUIP/PORT 25/07            BATCH,MYFILE,PRINT,CR

COMMAND-ASSETS

ASSETS OF CR AT 11.53.15.   --Use this identifier in command:
EQUIP/PORT     25/07          BATCH,MYFILE,PRINT,CR
FILE QUOTA
                  20
FILES IN USE       1
MAX FL        077700
TIME LIMIT      0500
CP TIME       91.595
SYS TIME     143.090
•
•
•
COMMAND-
```

NOTE

The receiving terminal has an identifier that is different from the sending terminal.

After you enter PRINT,x, INTERCOM prompts you as follows:

```
TYPE FILE ID
```

Enter a 1- to 4-character alphanumeric identifier of the batched file. The operating system creates a banner page with the following identifier printed on it.

Iidentsn

I	This file originated from INTERCOM
ident	1- to 4-character alphanumeric file identifier, left-justified with zero fill
sn	System-generated sequence number

If the receiving terminal is a dial-up terminal (or if the user id was specified), the file appears under the category REMOTE OUTPUT when you issue the FILES command. The receiver must use the BATCH command to change the file from the REMOTE OUTPUT to the LOCAL category to display the file.

When using the BATCH command to send a file to a terminal that is not logged in, you should:

Use the disposition PRINT,MINE in the BATCH command.

Execute the LOGOUT command at the sending terminal.

Go to the receiving terminal and LOGIN with the same user name and password previously used at the sending terminal. You are assigned the same identifier for USER-ID.

At completion of LOGIN, issue the FILES command to display the file name in the form Iidentsn as a REMOTE OUTPUT file.

Use the BATCH command with a LOCAL disposition to make the file available for the EDIT and LIST commands or to rewind and copy it (with the COPYSBF command) to a connected file.

During the time you are moving from one terminal to another, the file remains on mass storage at the central site with the USER-ID of the user who issued the BATCH command. You can go to any other terminal and make the file available by logging in with the same user name and password used originally.

A REMOTE OUTPUT file is not a permanent file.

A local file referenced in a BATCH command giving PRINT disposition no longer exists as a local file, unless that file was an attached permanent file.

LOCAL OPTION OF BATCH

The LOCAL disposition of BATCH is used to change a REMOTE OUTPUT file to a LOCAL file. Files with REMOTE OUTPUT status are not under immediate user control; they cannot be referenced in EDIT, RETURN, STORE, or most other commands. Only files with LOCAL status are available for such uses.

REMOTE OUTPUT files usually originate at a terminal that does not have a printed copy capability or from a terminal with a card reader attached. They can also originate from your own terminal as a result of a remote batch job execution.

Disposition to change a REMOTE OUTPUT file is simply:

```
LOCAL
```

RENAME OPTION OF BATCH

The RENAME disposition changes the name of an existing local file to some other name. The new name cannot duplicate the name of any existing local file, and must consist of 1-7 letters or digits beginning with a letter.

This disposition is simply:

```
     RENAME
```

System interactive prompting continues with:

```
     TYPE NEW FILE NAME –
```

If you receive a FILE NOT AVAILABLE diagnostic while trying to use RENAME, perhaps you have misspelled the old file name or are attempting to rename a file that is not in the category LOCAL.

RENAME changes the local file name of an attached permanent file. It does not, however, affect the permanent file name by which the file is known to the system.

EXAMPLES OF BATCH COMMANDS

The sequence in the following example shows several local and remote output files. You can monitor how the BATCH command changes their categories by using the FILES command.

As indicated by the results of the first FILES command, two LOCAL and one REMOTE OUTPUT files exist at your terminal. Subsequently, the BATCH command is used to prin't one of those LOCAL files, HUH, at your own terminal. Using this method, the file is sent from one category at your terminal to another. Note that the next FILES command no longer shows a local file named HUH but does show an additional REMOTE OUTPUT file INOW064. File INOW064 is the LOCAL file HUH renamed by the BATCH command. (The name NOW was entered in response to the BATCH command TYPE FILE ID request. The prefix I is added to identify the job as an INTERCOM job; 064 is suffixed with 0 as filler and 64 as a unique identification.)

The third series of commands demonstrates three features. First, the BATCH command is used to reverse the procedure just demonstrated; that is, a REMOTE OUTPUT file is made LOCAL. Notice that after the last FILES command, file IMINE63 no longer exists as a REMOTE OUTPUT file but rather as a LOCAL file.

Also demonstrated is the feature of the BATCH command which allows multiple files to be processed by one command. Note that when the BATCH command verb is entered without parameters, INTERCOM automatically repeats the interrogation cycle until END is entered.

The last BATCH entry sends the LOCAL file PRINTIT to your terminal as REMOTE OUTPUT file IDIOT7E. This procedure is no different than that demonstrated in the first example. However, notice that PRINTIT remains as a LOCAL file as well as a REMOTE OUTPUT file. This occurs because PRINTIT is an attached permanent file.

```
COMMAND- FILES
--LOCAL FILES--
  HUH       *PRINTIT
--REMOTE OUTPUT FILES--
IMINE63


COMMAND- BATCH,HUH
TYPE DISPOSITION-PRINT,CN
TYPE FILE ID-    NOW
COMMAND- FILES
--LOCAL FILES--
  *PRINTIT
--REMOTE OUTPUT FILES--
IMINE63    INOW064


COMMAND- BATCH
TYPE FILE NAME-   IMINE63
TYPE DISPOSITION-LOCAL
TYPE FILE NAME-   PRINTIT
TYPE DISPOSITION-PRINT,CN
TYPE FILE ID-     DIOT
TYPE FILE NAME-   END
COMMAND- FILES
--LOCAL FILES--
  IMINE63   *PRINTIT
--REMOTE OUTPUT FILES--
IDIOT7E    INOW064
```

BYE COMMAND

BYE is an EDITOR command. It must be the last command entered under EDITOR, since it causes an exit from EDITOR and a return to COMMAND mode.

WHEN IS BYE USED?

All terminal operations are complete and you are going to stop communication with the LOGOUT command.

The next series of operations requires INTERCOM, but not EDITOR, capabilities.

Beginning users will probably call BYE only when ending a terminal session.

Experienced users, however, often execute a series of operating system control statements or XEQ or PAGE commands. These commands can be entered through EDITOR, but quicker response occurs when they are entered from COMMAND mode. The user must weigh the time gained by leaving EDITOR against the time required to re-enter EDITOR for future file editing.

BYE FORMAT

BYE	abbreviated B
BYE,BYE	abbreviated B,B

Using BYE causes a check of edit file contents and a potential delay to preserve the edit file contents. BYE executes only if:

The current edit file has been referenced in a SAVE command.

The current edit file is empty.

The current edit file has not been modified.

If EDITOR determines that a new or modified file has not been saved, this message appears:

```
WARNING- EDIT FILE NOT SAVED
```

At this time the user can:

Use the SAVE command to preserve the edit file; or

Re-enter BYE to exit EDITOR.

Using BYE,BYE causes an immediate exit from EDITOR with no warning message even if the edit file has not been saved. The informative message

YOU HAVE AN EXISTING EDIT FILE

appears if you re-enter EDITOR without logging out.

CONNECT COMMAND

CONNECT is an INTERCOM command that equates a file with the display area of a terminal or with keyboard entry from a terminal. Each line of any connected output file is displayed as a program or utility writes to that file. Similarly, execution of any program that reads a connected input file is suspended until an input line is received from the terminal keyboard. The opposite of CONNECT is DISCONT, which disconnects a connected file.

WHEN IS CONNECT USED?

CONNECT is used mainly to display information at a terminal, especially for files that have no use after display. Typical uses are:

To examine a file to be produced during utility execution such as AUDIT or COPYSBF.

During debugging of a program; connect the output file to determine whether results are acceptable. If so, abort execution, disconnect the output file, and execute the program with output written to a file that can be preserved for future use.

To enter data into an executing program without first creating a separate local file with that data.

The RUN command connects files with names INPUT and OUTPUT every time it is executed. It is not necessary that a program use the connected files however.

Any local file may be connected. Although experienced programmers may connect an existing file and preserve its contents during connected operations, it usually is more convenient to work with a new file having no contents. Only this application is discussed below.

Connected files have these characteristics:

A dollar sign appears before the file name in the list of LOCAL files. The $ is not part of the file name.

A copy of information read or written while the file was connected does not exist on mass storage.

CONNECT FORMAT

The command that connects a file to the terminal cannot be abbreviated.

```
CONNECT,file
```

file	Name of file to be connected, 1-7 letters or digits beginning with a letter.

More than one file can be connected with a single command:

```
CONNECT,file1,file2, . . .
```

CONNECT may be used to determine the contents of an existing file named UNKNOWN:

```
COMMAND-CONNECT,SCRN
COMMAND-REWIND,UNKNOWN
COMMAND,COPYSBF,UNKNOWN,SCRN
UNKNOWN LINE 1
UNKNOWN LINE 2
UNKNOWN LINE 3
UNKNOWN LINE 4
COMMAND-
```

The COPYSBF command inserts a carriage control character before each line to cause single spaced lines. The lines from file UNKNOWN are displayed as they are copied. At the end of COPYSBF execution, SCRN has no information: all information has been transferred to the terminal and displayed.

If you try to edit SCRN, you receive this message.

```
ERR-SCRN CONNECTED TO TERMINAL
```

CREATE COMMAND

CREATE is an EDITOR command used to create a file line by line from keyboard entries. Once CREATE is entered, each entry terminated by a RETURN key is presumed to be a line of text for the evolving file. Until CREATE is terminated, other EDITOR or INTERCOM commands are not recognized as such. CREATE does not terminate until the user enters an equals sign.

When CREATE is called:

Each line entered by the user acquires a system assigned line number.

The end of a file line is signaled by the RETURN key. As many as 510 characters can be entered in a single line.

A tab character defined through the FORMAT command can be used to position characters entered.

All characters entered at the keyboard are assumed to be part of the file until = is received as the only line character.

WHEN IS CREATE USED?

The CREATE command is used to construct a file. This file may contain:

A program to be executed through the RUN command.

Data to be used by a program

Experienced programmers also may use CREATE to form a file of further instructions such as a directive file for UPDATE use or a complete job deck.

CREATE FORMAT

The command that initiates construction of a new file in the temporary edit file work area has three optional parameters.

CREATE,start,increment,SUP   abbreviated C

start	Number of first line in the edit file. Default value is 100.
increment	Number added to the last line to determine value of next line number. Default value is 10.
SUP	abbreviated S Suppresses prompting of user response through display of line numbers.

The numerical parameters are positional: if an increment is to be stated, the starting line number must be given.

System response to CREATE is to prompt user input of the first line by displaying the number of that line:

```
..CREATE,1,5
     1=USER TEXT GOES HERE FOLLOWED BY RETURN KEY
     6=
```

When the SUP parameter is used, response is:

```
..CREATE,SUP,1,4
ENTER LINES
```

In both instances, the user enters lines until the file has been constructed. To terminate CREATE, enter an equals sign as the only character in the line.

```
..CREATE,15,6
    15=FIRST USER LINE
    21=SECOND USER LINE
    27==
..
```

If a tab character has been defined by the FORMAT command or default FORMAT setting, that character acts similarly to a SKIP key on a keypunch or a TAB key on a typewriter. Each time the tab character is used in the line, the next input character is located at the next defined tab stop. If no more tab stops are defined, however, the tab character becomes part of the input line.

Assume a file of data statements is to be constructed. The FD in a COBOL program is:

```
01   DATAIN.
     03   NUMB1 PIC 9(4).
     03   FILLER PIC XXX.
     03   NUMB2 PIC 9(7).
     03   FILLER PIC XXXX.
     03   NUMB3 PIC 9(10).
```

Data cards should be formatted with tab stops at positions 8, 19

```
..FORMAT,8,19
..CREATE,SUP
ENTER LINES
1111   2222222    3333333333
4444;5555555;6666666666
7777;8888888;9999999999;
=
..LIST,100,LAST
   100=1111   2222222    3333333333
   110=4444   5555555    6666666666
   120=7777   8888888    9999999999;
..
```

Since no more tab stops are defined, the last semicolon becomes part of text

A common mistake beginners often make, particularly when SUP has been used, is forgetting to exit from CREATE with an equals sign. CREATE continues accepting entries with no visible results:

```
..CREATE,1,1,SUP
ENTER LINES
FORMAT,TAB=*,50
FIRST LINE TEXT*AFTER TAB TO 50
ANOTHER LINE
LAST LINE USER WANTS* STAR CAUSES TAB TO 50.
SAVE,NEWFILE
STORE,NEWFILE,NOUN
BYE,BYE
LOGOUT
LOGOUT
HELP, THIS ISNT STOPPING
```

An abort through %A, or the single =, will terminate CREATE at this point.

To recover from the previous situation after an abort, the command lines can be deleted; but since the FORMAT command appeared after CREATE, it is not recognized and the lines must be re-entered:

```
..FORMAT,TAB=*,50
..CREATE,1,1
WARNING- EDIT FILE NOT SAVED
..CREATE,1,1
      1=FIRST LINE*AFTER TAB TO 50
      2=ANOTHER LINE
      3=LAST LINE USER WANTS*ASTERISK CAUSES TAB TO 50
      4==
```

DELETE COMMAND

With this EDITOR command, one or more lines in the edit file can be deleted. The lines to be eliminated can be specified explicitly by number or implicitly by a search criteria. An entire line, not just part of a line, is affected by DELETE.

Many of the options of DELETE are the same as for SAVE, LIST, and the text replacement command.

WHEN IS DELETE USED?

To eliminate extraneous lines in the file, such as those arising from input typing errors.

To remove program statements used to debug a program.

To empty the temporary edit file in preparation for new file construction.

DELETE is not needed before replacing a single line. By using a command in the following format, the entire specified line is rewritten.

```
linenum=text
```

The text replacement command, /oldtext/=/newtext/, also can be used to correct a line without deleting the incorrect line first.

DELETE FORMAT

The simplest form of DELETE specifies only the lines to be eliminated from the edit file:

DELETE,range	abbreviated D,range

The range parameter identifying the lines to be deleted may have one of these forms:

ALL	abbreviated A The entire edit file is deleted.
LAST	abbreviated L Only the last line in the file is deleted.
line number	Only this line is deleted.
line-1,line-2	Lines within this range are deleted.
line-1,LAST	Line-1 through the last line of the edit file are deleted.

SEARCH CRITERIA OPTION OF DELETE

Indirect specification of a line is possible by specifying search criteria. Only lines within the range given are searched, and then deleted if they meet qualifications.

/text/	Only lines containing this text string are deleted. /text/ may be further qualified by the (column) and UNIT options.
(column)	/text/ character string must begin in this column to satisfy the search criteria.
(column-1,column-2)	/text/ character string must be contained within the column range specified.
UNIT	/text/ must not be preceded or followed immediately by a letter or digit.

VETO OPTION OF DELETE

Lines to be deleted should be specified precisely, since no restoration of deleted lines is possible except by re-entering each deleted line individually.

The VETO option can be used to check each line before it is deleted. Option format is the word:

VETO	abbreviated V

When VETO is used, each line that meets specified search criteria is displayed before deletion proceeds. After examining the line, you can enter YES, NO, or CONTINUE to accept or reject deletion of that particular line.

YES	Causes line to be deleted; abbreviated Y
NO	Retains this line; abbreviated N
CONTINUE	Deletes this line and all others which meet search criteria in range remaining. Remaining deletions are not displayed; abbreviated C.

The NO response affects only the line displayed; it does not terminate DELETE. To bypass all other lines without deletion, terminate the command with %A.

DISCARD COMMAND

DISCARD is an INTERCOM command used to eliminate a local or permanent file. Its primary function is to delete a permanent file, but it can be used with an unconnected local file except one with the name INPUT.

WHEN IS DISCARD USED?

To eliminate a file made permanent with a STORE command.

To eliminate any unconnected local file from your list of files.

DISCARD FORMAT

DISCARD has two formats; the command has no abbreviation or optional parameters.

```
DISCARD,file

DISCARD,permname,owner
```

The first format is used to eliminate any local file, including an attached permanent file. The second format is used to eliminate a permanent file that has not been made local by FETCH.

file	Name of local file
permname	File name made permanent by STORE
owner	Name used as owner identifier for STORE

Only one file name can be used. In response to DISCARD, the file ceases to exist as a LOCAL file and the permanent file is purged.

For permanent files, either of the two following sequences eliminates a file from the system Assume a file was made permanent by the command: STORE,SAMPLE,MINE.

```
••FETCH,SAMPLE,MINE
••DISCARD,SAMPLE
••
```

or

```
••DISCARD,SAMPLE,MINE
••
```

DIAGNOSTIC FROM DISCARD USE

An error in entering this command produces one of the following messages. Re-enter the command correctly.

ERR – CANT FIND FILE file

File is neither local nor attached permanent file, nor a permanent file with the correct owner id specified. Make certain you have spelled the file name correctly.

When a permanent file cannot be located, enter the AUDIT command with your owner identification. Perhaps the file was identified with another name on STORE. Alternately, the expiration date for a file may have been reached, and the file may no longer exist.

ERR – FILE NAME REQUIRED

You have neglected to specify a file name.

ERR – FILE NAME MUST BE ALPHANUM,<8 CHAR, 1ST CHAR A-Z

You did not enter a legal file name.

ERR – TOO MANY PARAMETERS

DISCARD,permfile,owner	Cannot be used to discard a local file.

CANNOT ROUTE INPUT FILE

You cannot discard a connected file. Use DISCONT command, then DISCARD. A similar message also appears if you are referencing the file named INPUT. To eliminate INPUT from the list of local files, you must rename it through the BATCH command and then DISCARD the new file name.

EDIT COMMAND

This EDITOR command transfers an existing local file with coded data to the temporary edit file in preparation for listing or modifying it. The reverse of EDIT is SAVE, which copies the edit file contents to a local file.

WHEN IS EDIT USED?

EDIT must be used before any of these commands if an existing local file is to be updated:

ADD	LIST	/oldtext/=/newtext/	linenum=text
DELETE	RESEQ	SAVE	

EDIT is not used before the CREATE command which constructs a new file.

Connected files cannot be transferred to the edit file.

EDIT FORMAT

The EDIT command must identify the local file to be placed in the edit file area. The optional parameter, SEQ, is required if the local file does not have ascending line numbers.

EDIT.file,SEQ	Abbreviated E,file,S
file	Name of local file with character contents
SEQ	Optional sequencing call. Line numbers begin with 100 and increment by 10

In response to EDIT, the named file is rewound and copied to the edit file work area.

If another file already occupying the edit file has not been saved or has been modified since it was last saved, a warning message is issued rather than destroying the file. To preserve the existing file, use the SAVE command; to destroy the existing file re-enter the EDIT command.

An alternative to re-entering EDIT, when you know you no longer need the current file is:

```
••DELETE,ALL   deletes all lines in current file
••EDIT,CALC,SEQ
••
```

If the current edit file has already been saved, or is an existing file that has not been modified, EDIT executes immediately.

The file to be transferred to the edit file can consists of a multi-file set. For instance:

```
••REWIND,TENKEYD
••REWIND,INFILE
••COPY,TENKEYD,NEWFILE
••COPY,INFILE,NEWFILE
••EDIT,NEWFILE
••LIST,ALL
   1=IDENTIFICATION DIVISION.
   2=PROGRAM-ID. TENKEYD.
•
•
•
```

```
   54=STOP RUN.
   55=*EOR
   56=*EOF
  100=000123
  110=000123
•
•
•
  220=        T
  230=*EOR
  240=*EOF
  250=*EOF
••
```

The character string *EOF becomes part of the edit file when EDITOR encounters the end-of-file indicator written by COPY. Similarly, an *EOR character string would appear for a logical record indicator.

Each line in the edit file must have a unique number; numbers must be in ascending order. Since the two files copied to NEWFILE did not have overlapping numbers, SEQ was not needed on the EDIT command.

An attempt to execute EDIT with a file not having numbers produces an error message. The present edit file work area is destroyed once EDIT command execution begins, as illustrated by:

```
••SAVE,FILE,NOSEQ
••EDIT,FILE
ERR- LINE NUMBERS OUT OF SEQUENCE
••SAVE,FILE
ERR- NO INFORMATION IN EDIT FILE
```

Either of the following is correct:

```
••SAVE,ROBIN NO SEQ        ••SAVE,CARD
••EDIT,ROBIN,SEQ           ••EDIT CARD
```

New line numbers can be added to existing line numbers. Since an EDIT command with SEQ assumes no line numbers exist, EDITOR treats actual numbers as part of individual lines. If you SAVE a file with line numbers and then call for EDIT with SEQ, a list of the file shows extraneous characters at the end of each line. These characters are the original line numbers, which in this example began at 100 and incremented by 10. This situation can be avoided by omitting the SEQ parameter when EDIT is used. If the file was saved without numbers, a diagnostic is issued when the EDIT command is given.

```
..SAVE,FINCH
..EDIT,FINCH,SEQ
..LIST,100,LAST
  100=THIS IS THE CONTENTS OF FILE A
      000100 <------------------------------ line numbers from
  110=*EOR                                   SAVE command
  120=*EOF
  130=THIS IS THE CONTENTS OF FILE B
      000130
..
  line numbers from
  EDIT command
```

A RESEQ command could change 100 to 500, for example, but could not change the 000100 at the end of the first line.

Lines of up to 510 characters can be transferred to the edit file. Characters beyond 510 are not moved but this message is displayed:

LINES > 510 CHARS WERE TRUNCATED

The above message may appear when you expect 72-character lines, if the named file contains binary data. First verify that the correct file name was used. EDITOR determines the end of a line in a local file by searching for a special indicator written at the end of every line with character data. If the line does not have character data, no indicator exists; EDITOR searches continuously, however, and can read an entire file in its search.

Any of the following messages indicates an error in the command entry. Check for misspellings and omitted separators.

ERR – CANT FIND FILE filename

ERR – FILE NAME MUST BE ALPHANUM, < 8 CHAR, 1ST CHAR A–Z

ERR – FILE NAME REQUIRED

ERR – PARAM n: UNRECOGNIZABLE PARAMETER

ERR – RESERVED FILE NAME

EDITOR COMMAND

EDITOR is the INTERCOM command used to call the file creating and editing facility of INTERCOM.

WHEN IS EDITOR USED?

A file is to be entered statement by statement.

An existing file is to be changed.

EDITOR must be called before any of these commands can be used.

ADD	DELETE	RESEQ	/oldtext/ = /newtext/
BYE	EDIT	RUN	linenum=text
CREATE	FORMAT	SAVE	LIST

EDITOR FORMAT

EDITOR

No optional parameters exist. The command cannot be abbreviated.

The system responds with two dots at the left. The appearance of these dots, rather than the word COMMAND–, after a user entry, indicates that EDITOR commands are valid.

Once called, EDITOR is available until the user enters the BYE command. An abort through a user entry of %A terminates the current command but not EDITOR itself. Remember, if output must be suspended, press ESC.

During EDITOR use, any INTERCOM command described in this manual can be entered except:

EDITOR LOGOUT LOGIN

FETCH COMMAND

FETCH is an INTERCOM command that retrieves a file made permanent by the STORE command.

WHEN IS FETCH USED?

To access a permanent file.

Execution results in the file being added to your list of local files. Attached permanent files are identified by an * preceding the file name.

FETCH FORMAT

FETCH format depends on installation procedures: the owner parameter may not be required. Any differences are the same for STORE and DISCARD. Follow any local instructions.

The word FETCH cannot be abbreviated. Standard format is:

FETCH,file,owner

file	Name by which file was made permanent with a STORE command. 1-7 letters or digits beginning with a letter.
owner	Identification of the file owner used in the STORE command. 1-9 letters or digits.

Successful attach of the file can be verified by the FILES command.

```
••FILES
NONE
••FETCH,GREBE,AUDUBON
••FILES
--LOCAL FILES--
   *GREBE
••
```

FILES COMMAND

FILES is an INTERCOM command used to obtain a list of all files associated with the terminal.

WHEN IS FILES USED?

Operations have been conducted on several files and you are not sure of the spelling of a particular file name.

You want to check whether a given file is connected or is a permanent file.

The message FILE QUOTA EXCEEDED has appeared and you must know the names of existing files to be returned before new files can be created.

In general, FILES can be used to refresh your memory before entering a command that references a new or existing file.

FILES FORMAT

FILES

No optional parameters exist. The command cannot be abbreviated.

SYSTEM RESPONSE TO FILES

When FILES is executed, a list of all files for your terminal appears:

–LOCAL FILES–	Files under immediate control
–REMOTE EXECUTING FILES– –REMOTE INPUT FILES–	Jobs named in BATCH command for transfer to central site
–REMOTE OUTPUT FILES– –REMOTE PUNCH FILES–	Results of BATCH use with your terminal identification

Headings appear only for the types of files to be listed.

LOCAL files may have a special character preceding the logical file name to show further status:

Prefix	**Meaning**
*	Attached permanent file
$	Connected file. Input comes from keyboard. Output is displayed.
none	File neither permanent nor connected.

Any change in status is reflected in this list.

At the beginning of a terminal session, FILES execution produces the response:

```
NONE
```

No files are associated with the terminal until the user enters a command to attach or create a file.

To remove a file from terminal control, use the RETURN or DISCARD commands.

DIAGNOSTIC FROM FILES USE

THE ABOVE LIST MAY BE INCOMPLETE

When a large number of files are in the queue, this message may be returned after the names listed under any of the file categories. In most cases, the list of terminal files is complete. Repetition of the FILES command may not change the information displayed.

FORMAT COMMAND

FORMAT is an EDITOR command that establishes the maximum line length, tab stops, and tabbing character for lines entered into the edit file from a terminal keyboard. FORMAT affects the following commands:

CREATE   linenum=text   ADD   /oldtext/=/newtext/   SAVE

WHEN IS FORMAT USED?

To change line length to a value required by a compiler

To change line length to a value required by an FD paragraph in a program

To make entering text easier by defining tab stops

To determine existing line length and tabs

When you become more experienced with program and line manipulation, you may want to use FORMAT to establish variable line lengths.

A default format exists, corresponding to the format of the language most frequently entered at an installation. To determine the default, enter the command:

```
. . FORMAT,SHOW
```

System response, if the default is FORTRAN, will be:

```
CH= 72 TAB CHAR=; TAB COL=    7
     ↗           ↗             ↗
     line length  tab character  tab stop
```

The response can be changed to COBOL settings as indicated below.

FORMAT FORMAT

The form of the command used most often contains a compiler language name.

```
FORMAT,COBOL
```

A line length of 72 characters, a tab stop at positions 8, 12, 16, 20, 24, and a tab character of a semicolon is thereby defined.

Other FORMAT compiler names allowed are:

	Line Length	Tab Character	Tab Stops
ALGOL	72	$	7, 10, 13, 16, 19
BASIC	72	;	None
COMPASS	72	;	11, 18, 36
FORTRAN	72	;	7

The three settings established by FORMAT can be set independently of compiler requirements in any order and value in any combination.

FORMAT, CH = length, TAB = char, stop, stop, . . .

Any number entered without a preceding CH = or TAB = is assumed to be a tab stop. Tab stops must be contiguous. For instance, FORMAT 5, TAB = %, 10 is an illegal statement.

CH=	abbreviated C= Maximum number of characters in line, 1-510.
TAB=	abbreviated T= or TA= Any character, except %. When used in a line entry, the tab character repositions the succeeding text character to the next defined tab stop.
stop	Any line position 0-510. A series of tab stops must be in ascending order. The tab stop position must not exceed line length set by the CH= parameter.

DIAGNOSTICS FROM FORMAT USE:

ERR- CH= MUST SPECIFY < 511

The longest line possible in the edit file is 510 characters.

ERR- TAB= MUST SPECIFY ONLY 1 CHAR

You probably have forgotten a delimiter after the tab character.

ERR- TABS TOO BIG OR OUT OF ORDER

Tab stops must be in ascending order.

LIST COMMAND

LIST is an EDITOR command that displays one or more lines of the edit file. The lines to be displayed can be specified directly by line numbers or indirectly by search criteria.

Many of the optional search criteria parameters of LIST are those of SAVE, DELETE, and the text replacement command.

WHEN IS LIST USED?

When you are creating or updating a file through EDITOR, use LIST to:

Display a line or an entire program or file so you can check for syntax, logic, or typing errors.

Determine line numbers of statements to be corrected.

Determine line numbers for statements to be added.

LIST works only with the edit file. Since a local file can be transferred to the edit file, you can use LIST to:

Examine the contents of a local file, such as the output from program execution.

LIST FORMAT

The simplest form of LIST specifies only the lines to be displayed.

LIST,range	abbreviated L,range

The range parameter identifies the lines to be displayed. It may have one of these forms:

ALL	abbreviated A The entire edit file is displayed.
LAST	abbreviated L Only the last line in the file is displayed.
line number	Only this line is displayed.
line-1,line-2	Lines encompassing this range are displayed.
line-1,LAST	Line-1 through the last line of the edit file are displayed.

If the range parameter is omitted, the last line previously displayed is repeated. The last line may have been the result of a LIST command or the VETO option of another EDITOR command.

The SUP optional parameter suppresses display of the edit file line numbers.

SUP	abbreviated S

The lines to be displayed can be specified by search criteria. Only lines within the range given with the first parameter will be searched and listed if they meet qualifications.

/text/	Only lines containing this text string are displayed. /text/ may be further qualified by the (column) and UNIT options.
(column)	/text/ character string must begin in this column to satisfy the search criteria.
(column-1,column-2)	/text/ character string must be contained within the column range specified.
UNIT	/text/ must not be preceded or followed immediately by a letter or number.

EXAMPLES OF LIST USE:

LIST,ALL,SUP	Display entire file without line numbers
LIST,ALL,/01/,UNIT	Display all lines containing 01 levels
LIST,200,300,/SELECT/	Display all SELECT statements in lines 200 through 300
LIST,ALL,(7),/*/	Display all comment statements
LIST,500,LAST,UNIT,/PERFORM-/	Display all lines in range 500 to end of file if the line has a PERFORM statement, but not user-names such as PERFORM-PARA.

RESEQ COMMAND

This EDITOR command renumbers the lines in the current edit file.

WHEN IS RESEQ USED?

Edit file line numbers must be in ascending order at all times. A new line cannot be inserted into the edit file unless an unused line number is available.

Resequence when line numbers are insufficient to allow insertions.

Resequence for your own convenience in referencing lines.

RESEQ FORMAT

The RESEQ command may be entered with or without numbering parameters:

RESEQ	abbreviated RE

RESEQ,start,increment

start	Line number for first line in file. Value may be 1-6 digits. Must be specified if an increment parameter is given.
increment	Value to be added to previous line number to determine next line number. Value may be 1-6 digits.

If RESEQ is entered without a starting line number and increment value, lines are numbered from 100 with increments of 10.

In response to RESEQ, the entire edit file is renumbered.

Assume a program in the edit file is not executing correctly. DISPLAY statements are to be inserted after a READ to verify data validity. The current file is:

```
221=              READ INFILE AT END GO TO CLOSE-PARA.
222=              MOVE CORR INREC TO STOREREC.
223=              IF FIELD1 GREATER THAN FIELD2 GO TO GR-PARA.
```

No new lines can be inserted between line 221 and 222. Resequence the file starting at 100 and incrementing by 5; then insert new lines.

```
..RESEQ,100,5
..LIST,ALL,/READ INFILE/
  120=             READ INFILE AT END GO TO CLOSE-PARA.
..121=;;DISPLAY "FILE READ" UPON HERE.
LIST,120,130
  120=             READ INFILE AT END GO TO CLOSE-PARA.
  121=             DISPLAY "FILE READ" UPON HERE.
  125=             MOVE CORR INREC TO STOREREC.
  130=             IF FIELD1 GREATER THAN FIELD2 GO TO GR-PARA.
..
```

Do not abort execution of RESEQ. An accidental abort must be followed by another RESEQ to re-establish edit file line number continuity.

DIAGNOSTICS FROM RESEQ ENTRY

Incorrect command format produces one of these diagnostics. RESEQ must be re-entered correctly.

ERR – PARAM n: ILLEGAL LINE NUMBER

Line number can be 1 through 999999

ERR – PARAM n: TOO MANY DIGITS

The starting line number and the increment value must not exceed 6 digits.

ERR – TOO MANY PARAMETERS

Correct format is: RESEQ,start,increment

ERR – NO INFORMATION IN EDIT FILE

RESEQ works only with the contents of the edit file. If you want to reassign line numbers for a local file, use the EDIT command first.

RETURN COMMAND

RETURN is an operating system command that releases a local file or several files from the terminal. Both the file and the file name cease to exist at the terminal.

WHEN IS RETURN USED?

A local file is no longer needed.

FILE QUOTA EXCEEDED message has appeared.

An attached permanent file is to be retained as a permanent file but released from the terminal.

RETURN affects only local files. An attached permanent file reverts to permanent file status only. A DISCARD command must be used to eliminate completely a permanent file.

A connected file must be disconnected before use in RETURN.

RETURN FORMAT

One or more files can be released. RETURN has no abbreviation. Comma separators should be used.

```
RETURN,file
RETURN,file,file, . . .
```

file	Name of file to be released

Special name files such as OUTPUT have meaning to the operating system. Releasing them from the terminal by RETURN is equivalent to a request for particular processing. A file with the name PUNCH, for example, is punched at the central site when released from the terminal.

Do not use RETURN with files named OUTPUT, PUNCH, P80C, PUNCHB, PRINT, or INPUT. Use the DISCARD command for all except INPUT, which must be renamed through BATCH before use in RETURN.

The list of your LOCAL files changes when RETURN is used:

```
..FILES
--LOCAL FILES--
   BB       AA       CC       DD
..RETURN,BB,CC
..FILES
--LOCAL FILES--
   AA       DD
..
```

No diagnostic is issued if the file name entered does not match the name of a local file. Consequently, omitting a comma between two file names does not produce a diagnostic message, but neither does it execute as desired. For example, files FF, AA, and DD exist:

```
..REWIND,FF
..RETURN,AA FF,DD
..FILES
--LOCAL FILES--
   AA      FF
..
```

Files DD and AAFF are returned, not AA and FF as expected.

DIAGNOSTICS FROM RETURN

RETURN, REWIND OR UNLOAD MUST HAVE AT LEAST ONE PARAMETER

You have omitted a file name

UNLOAD NOT ALLOWED ON INPUT

The RETURN command cannot be used with a file by the name of INPUT. To release INPUT, use two commands:

```
BATCH,INPUT,RENAME,NEW

RETURN,NEW
```

REWIND COMMAND

REWIND is an operating system command that can be used through INTERCOM. It causes the named file to be rewound to beginning of information. A connected file must be disconnected before it is rewound.

If the file named on a REWIND does not exist as a local file, using that name with REWIND is sufficient to create a file with that name. No information is contained in the file, but the file exists.

WHEN IS REWIND USED?

To position a file to beginning of information.

To prepare a file for use after a COPY.†

To create a scratch file with no data while you are experimenting with INTERCOM command use.

For the INTERCOM applications described in this manual, usually it is not necessary to specify REWIND. Execution of most EDITOR and INTERCOM file commands implies a rewind. For example, SAVE and EDIT rewind the referenced local file. RUN rewinds the files INPUT, OUTPUT, and LGO before sending the program in the edit file to the compiler. Similarly, BATCH rewinds the file before sending it to another terminal.

Users executing COBOL programs must rewind files with the REWIND command if the program uses the WITH NO REWIND option of the OPEN and CLOSE statements.

REWIND FORMAT

A REWIND command can specify a single file or several files. REWIND cannot be abbreviated.

```
REWIND,file

REWIND,file1,file2,file3, . . . , filen
```

file	Name of file to be rewound.

The maximum number of files that can be rewound by a single command is limited only by the fact that no command can exceed 80 characters, including separators and a final period that is supplied by INTERCOM if the user omits it.

RUN COMMAND

This EDITOR command compiles and executes a program in the edit file. It also can be used to compile and execute any other local file. During execution, local files referenced by the program can be read and new local files created.

†COPY is a very useful utility routine. However, it does not rewind files either before or after execution; thus, a user initiated rewind before and after use may be necessary. For more information concerning COPY, refer to the INTERCOM and operating system reference manuals.

WHEN IS RUN USED?

To compile and execute a source language program.

To check out program or subroutine accuracy by compiling, but not executing, statements.

Learn acceptable COBOL format by compiling test statements that do not form a complete program.

RUN FORMAT

The minimum command that compiles and executes a COBOL program residing in the temporary edit file is:

RUN,COBOL	Abbreviated RU,COB
COBOL	Required name for identifying the COBOL compiler

Optional parameters can be added singly or together in any order:

FILE=lfn	Local file named, not the edit file, is passed to the compiler
NOEX	Causes compilation; but not execution
RUN,COBOL,FILE=lfn,NOEX	Abbreviated RU,COB,F=lfn,N

System response to the minimum command format is:

EDITOR creates a local file copy of the edit file which is then rewound, as is file OUTPUT and LGO.

COBOL compiler compiles the program and writes the object code to file LGO.

Files INPUT and OUTPUT are connected to the terminal.

File LGO is executed.

Variations in response occur when the compiled program has errors or the NOEX option is selected.

Executing the FILES command after RUN,COBOL shows three files that may not have existed previously:

```
. . FILES

$INPUT    $OUTPUT    LGO
```

Files INPUT and OUTPUT are connected to the terminal, as indicated by the $ preceding the file names. Any statement in the program reading a file with the name INPUT stops execution and waits for entry from the terminal of information to be read. Similarly, any statement writing to a file with the name OUTPUT displays information at the terminal as it is generated.

INPUT and OUTPUT are always connected if the program is to be executed; it does not matter whether or not they are referenced in the program. The INTERCOM command DISCONT entered before RUN does not prevent reconnection during RUN execution.

If execution of your program produces any errors, you will receive the following diagnostics as a result of RUN execution:

```
..RUN,COBOL
 COMPILING DATARD
  000 E AND      T/U DIAGNOSTICS ISSUED
    054000B SCM USED
      .258 CP SECONDS COMPILATION TIME
END COBOL
AFILE - OPEN OLD ON A NON-EXISTING FILE
 SOURCE LINE NUMBER 00023
 FATAL COBOL ERROR - RUN TERMINATED
```

Compilation errors, if any, would be displayed before the execution time errors.

NOEX OPTION OF RUN

Execution is suppressed when NOEX is used as a RUN parameter. Use NOEX the first time you compile a program unless you are certain no format errors exist.

Without NOEX, the system calls the loading routine and attempts to load the LGO file generated during compilation. A fatal error diagnosed by the compiler, however, sets a fatal error flag at the beginning of the LGO file. If the loader detects the fatal flag, loading is stopped and a message is output to the terminal. Meanwhile the loader call is wasted.

Format of the option is:

NOEX	Abbreviated N

The use of the NOEX option creates the file LGO containing the compiled program and connects the file OUTPUT. It does not create and connect the file INPUT.

```
..RUN,COBOL NOEX
 COMPILING TENKEYD
  000 E AND      T/U DIAGNOSTICS ISSUED
    053377B SCM USED
      .368 CP SECONDS COMPILATION TIME
END COBOL
..FILES
--LOCAL FILES--
  $OUTPUT      LGO
..
```

If you have compilation errors, the file OUTPUT is not connected.

At the end of RUN with NOEX, the compiled program exists on the file LGO and can be executed by:

```
..LGO
```

The system accepts this command as a call to rewind, load, and execute a file named LGO which has binary data. As long as OUTPUT is connected, program DISPLAY statements or WRITE statements to a file equivalenced with OUTPUT are displayed as a result of LGO.

```
..CONNECT,OUTPUT
..LGO
THIS PROGRAM RUN THROUGH EDITOR
..
```

FILE OPTION OF RUN

To compile and execute a COBOL program not in the edit file, use the FILE option. The file named can be any local file. The file must be positioned at the beginning; if in doubt, rewind.

Contents of the file must be a source language program, not binary output such as produced on file LGO.

Parameter format is:

FILE=name Abbreviated F=lfn

Spaces are not allowed within the parameter.

Either of the following sequences produces the same results:

```
..EDIT,SOURCE,SEQ                    ..RUN,COB,FILE=SOURCE
..RUN,COB
```

SAVE COMMAND

SAVE is an EDITOR command. It copies the current contents of the edit file into a local file. SAVE neither changes nor destroys the current edit file. Rather, it copies the temporary edit file to the file named in the SAVE command.

WHEN IS SAVED USED?

Information entered under the CREATE command of EDITOR is to be preserved for later use.

A file has been edited, and the changed file is to be preserved.

The edit file is to be stored as a permanent file.

The edit file is to be copied or otherwise referenced by name.

SAVE does not make a file permanent or preserve it from one day to the next. The STORE command, not SAVE, is used for these purposes. SAVE preserves a file only for the current terminal session. Once a terminal has been logged out, any file not made permanent is lost.

SAVE FORMAT

The simplest form of the SAVE command is:

SAVE,filename	Abbreviated S,filename

Filename may be any combination of 1–7 letters or digits beginning with a letter. It must not be the same name as any other local file now existing at your terminal, unless either the MERGE or OVERWRITE option (explained below) is used also.

The names of existing files can be learned by using the FILES command.

In response to a SAVE command, the entire edit file is copied to a local file. Logically, these steps occur within EDITOR:

1. Request is made for a logical file on a permanent file device. The permanent file device request is made in anticipation of future STORE command. Files saved through EDITOR are always written to permanent file devices without user action.

2. The named local file is rewound.

3. The edit file is copied to the named file. Numbers assigned to each line are copied also; they appear as the last 6 characters of each line. The NOSEQ option stops the line number copy.

4. A system end-of-record, but not an end-of-file, terminator is written after all user data is copied. (See the EDIT command discussion for implications of the end-of-record mark.)

5. The named local file is rewound.

Length of the lines saved is determined by the FORMAT command currently in effect. Under the COBOL 72-character line length, 72 user data characters and 6 system supplied digits (line number) would exist for each line in the edit file.

Options of the SAVE command allow the following variations of system response:

OVERWRITE	Rewrites and consequently destroys, an existing file with the same name.
NOSEQ	Suppresses copy of line numbers to local file.
MERGE	Writes at present position of an existing file with the same name; inhibits rewind before and after writing.
VETO	Allows user to approve or disapprove saving of individual lines.

ALL, line numbers, and LAST save selected parts rather than the entire edit file.

/text/,column numbers, and UNIT save only those edit file lines that meet text search criteria.

These options can be used in any combination and in any order.

In most instances, only the NOSEQ and OVERWRITE parameters are used. The selective SAVE parameters that specify a range of line numbers, search criteria, and VETO options, are the same as for the DELETE, LIST, and /oldtext/=/newtext/ commands.

Only NOSEQ, OVERWRITE, and MERGE options are discussed below.

OVERWRITE OPTION OF SAVE

Every file associated with a terminal must have a unique name. An attempt to make a second file with the same name results in the following diagnostic, but not a new file.

```
ERR - - filename ALREADY EXISTS
```

OVERWRITE allows the user to replace or rewrite a file with a given name by replacing the existing file with the contents of the current edit file. The existing file is destroyed.

Format of the option is simply the parameter name. It can appear anywhere after the file name.

OVERWRITE	Abbreviated O

OVERWRITE is used commonly when a program or data in the edit file has been saved with a specific name, and then the edit file has been modified. To make the named file correspond to the modified file, use OVERWRITE.

An alternative to OVERWRITE is a sequence of operations that would independently release a file or one that would simply save the modified file with a different name. By using the OVERWRITE option of SAVE, an existing file, which is no longer valid, can be eliminated as well as a new local file created. These two sequences result in identical information for file IN.

```
..SAVE IN                             ..SAVE IN
(COMMANDS TO CHANGE EDIT FILE)        (COMMANDS TO CHANGE EDIT FILE)
..RETURN,IN                           ..SAVE,IN,OVERWRITE
..SAVE,IN
```

Overwrite also occurs without an OVERWRITE parameter if an existing file is saved without a MERGE parameter and is then referenced with a MERGE parameter.

Permanent files attached to your terminal cannot be overwritten. If a file listed by a FILES command execution with a preceding * is referenced with OVERWRITE, this error message appears:

```
ERR- OVERWRITE ILLEGAL ON PERM FILE
```

After such a message appears, either of the sequences shown below can be used:

To use a new permanent file name and eliminate old file

```
..SAVE,NEWNAME
..STORE,NEWNAME,OWNER
..DISCARD,OLDNAME†
```

To change contents but keep file name

```
..DISCARD,OLDNAME†
..SAVE,OLDNAME
..STORE,OLDNAME,OWNER
```

MERGE OPTION OF SAVE

A new file can be constructed by selectively saving parts of one or more existing files with the MERGE option. Lines to be saved can be identified by either a line number or a search criteria.

The merge function is accomplished by adding a parameter to the SAVE command each time a specific file is referenced:

MERGE	Abbreviated M

For example, assume that you have an existing COBOL program in the edit file and an existing subroutine ERREXIT on another file. Use the MERGE parameter to make the subroutine part of the program file named BOTH for execution through the RUN command.

(COBOL program in edit file)

```
. . SAVE,BOTH,MERGE,NOSEQ          Save main program without line numbers

. . EDIT,ERREXIT,SEQ               Move subroutine to edit file area

. . SAVE,BOTH,MERGE,NOSEQ          Add subroutine without line numbers to main program

. . EDIT,BOTH,SEQ                  Move merged file to edit file area

. . LIST,ALL/*EOR/                 Look for *EOR

    160 = *EOR

. . DELETE,160                     Delete *EOR

. . RUN,COB                        Execute program
```

†If the permanent file to be discarded has been attached, the owner's name must not be specified; including the name causes the diagnostic: ERR- TOO MANY PARAMETERS.

Line 160 shows *EOR when the merged file is listed. This end-of-record indicator is written by EDITOR after the last line of any item copied by SAVE execution. *EOR cannot be inhibited.

Since, in the preceding example, an end-of-record indicator would prohibit successful execution, the line containing *EOR must be deleted. In other instances, such as creating a batch job deck by merging a control statement record with a file containing a source language program, the end-of-record would be required.

The simplest way to delete any unwanted end-of-record indicators uses the DELETE command with search criteria. Otherwise, the user must list the file to determine the specific line number to be deleted.

. . EDIT,BOTH,SEQ	Loads merged file into the edit file
. . DELETE,ALL,/*EOR/,(1)	Search for and delete all lines with characters *EOR beginning in column 1

If you know the approximate size of two merged files, restrict the search for a line to be deleted to a reasonable range to decrease execution time. If a file with approximately 200 statements is merged with a second file of 50 statements for instance, assuming default edit file sequence numbers.

```
..DELETE,2000,2200,/*EOR/,(1)
```

MERGE should be included on the first SAVE of a file under construction if information is to be written at the end. Loss of information may result if MERGE is omitted, for the following reasons: SAVE without a MERGE parameter concludes execution with a rewind of the named file, so current file position becomes the beginning of information. Any time MERGE is used, the new material is written starting at the current file position on the assumption that the preceding MERGE left the file positioned at end of information. Consequently, a default OVERWRITE occurs if MERGE is not used on the first file reference.

The two following sequences illustrate an unsuccessful and a successful MERGE.

. . EDIT,PROG . . SAVE,NEW	File PROG is copied to NEW and NEW is rewound.
. . EDIT,SUB . . SAVE,NEW,MERGE	SUB is copied at the beginning of NEW, thereby destroying NEW.
. . EDIT,PROG . . SAVE,NEW,MERGE	File PROG is copied to NEW and NEW is left positioned at end.
. . EDIT,SUB . . SAVE,NEW,MERGE	SUB is copied after PROG, and NEW is left positioned at end.

The NOSEQ parameter is not required with MERGE, but MERGE and NOSEQ often are used together, as discussed under the NOSEQ option. The presence or absence of line sequence numbers has no effect on the file as saved. An EDIT of the merged file, however, requires ascending sequence numbers.

The following commands illustarte a common sequencing error when MERGE is used.

```
..EDIT,PROG,SEQ
..SAVE,NEW,MERGE
..EDIT,SUB,SEQ
..SAVE,NEW,MERGE
..EDIT,NEW

ERR- LINE NUMBERS OUT OF SEQUENCE
```

In this instance, both the files PROG and SUB had default line numbering, beginning with 100 and incrementing by 10; line numbers on the merged file NEW are duplicated and cannot be used for edit file purposes. Add the NOSEQ parameter to each SAVE of file NEW and subsequently use EDIT with the SEQ parameter.

Normally, the first SAVE command with MERGE is the first reference to a particular file name: any file existing as a local file can be merged by first using the EDIT command to transfer the file into the edit file, then executing a SAVE command with MERGE parameter. If the local file is to be retained in its entirety, however, information can be merged at the end as follows:

Assume a large existing local file named BIGFILE. The current edit file contains information to be merged at the end. To add the new information without first moving BIGFILE to the edit file, the control statement SKIPF can be used; without the 7777 parameter, BIGFILE would have been positioned after the first *EOR encountered.

. . SKIPF,BIGFILE,7777	Move current position to end of existing file.
. . SAVE,BIGFILE,MERGE	

The resulting file still has the *EOR at the end of the original BIGFILE data.

NOSEQ OPTION OF SAVE

Each line of the edit file has a unique line number associated with it. SAVE execution preserves these numbers, unless the NOSEQ option is selected to suppress the numbers.

Saving a file with line numbers does not interfere with compiler action, under FORMAT, COBOL conditions, since the compiler uses only 72 characters of input and the line numbers appear in columns 73-78. Under other FORMAT settings, however, or when data for program execution is being generated, line numbers may interfere with the interpretation of the file.

This option can appear any place after the file name of the SAVE command.

NOSEQ	Abbreviated N

NOSEQ is frequently used with the MERGE option when parts of several files are being written as one file, since the line numbers of any file to be edited must be in ascending sequential order.

For example, an existing file has line numbers 100-500. It is to be merged into the current edit file which has line numbers 100-300 and then edited. In the following sequence of commands, the user must know the highest line number in the current edit file and call for renumbering of the second file accordingly.

```
..SAVE,NEWFILE,MERGE
..EDIT,OLDFILE
..RESEQ,310,10
..SAVE,NEWFILE,MERGE
..EDIT,NEWFILE
..
```

The same results could be accomplished by one less user command and without the need to know any line numbers in the current edit file.

```
..SAVE,NEWFILE,MERGE,NOSEQ
..EDIT,OLDFILE
..SAVE,NEWFILE,MERGE,NOSEQ
..EDIT,NEWFILE,SEQ
..
```

STORE COMMAND

STORE is an INTERCOM command that makes a local file a permanent file. Permanent files, unlike local files, are preserved at the central site when you enter a LOGOUT command. They can be accessed in the future by entering a FETCH command.

WHEN IS STORE USED?

To make a file permanent use STORE when:

You are going to use the same program at successive terminal sessions.

Many users are going to access the same program at different times.

A message from the central site says INTERCOM is going down for a short time, and you have not finished your tasks.

Permanent files tie up system resources. Do not use them needlessly.

Experienced users often use the control statement CATALOG in place of STORE. STORE is adequate and simpler to use, but your instructor may have all your class files cataloged with one name.

STORE FORMAT

Normally, STORE format is as follows. No abbreviations or optional parameters exist. At some installations, another format is required.

STORE,file,owner

file	Name of any local file
owner	Identification of the file owner. Usually this is your name restricted to 1-9 letters.

The file name and owner are both required to use the file at any future time.

Use the FETCH command to make a permanent file local and the DISCARD command to destroy the file.

Assume a file made permanent with: STORE,EXAMP,SMITH

Access file FETCH,EXAMP,SMITH
or
Destroy file DISCARD,EXAMP,SMITH

Only local files can be made permanent. If you have created a file using EDITOR, you must first make the temporary edit file a local file before you can make it permanent.

```
..CREATE,SUP
   (FILE INFORMATION HERE)
..SAVE,SAMPLE,NOSEQ
..STORE,SAMPLE,JONES
..
```

A file you have just made permanent is still available to you as a local file. Execution of the FILES command shows the name with a preceding asterisk, showing it to be an attached permanent file.

```
..SAVE,SAMPLE,NOSEQ
..FILES
--LOCAL FILES--
   SAMPLE       UNPERM       OTHER
..STORE,SAMPLE,JONES
..FILES
--LOCAL FILES--
   *SAMPLE       UNPERM       OTHER
..
```

TEACH COMMAND

TEACH is a utility that summarizes INTERCOM operation, commands, and syntax. It consists of a series of displays about various topics.

WHEN IS TEACH USED?

The main TEACH display lists five items that can be further examined:

```
        TEACH
      IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
    TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

    HOW TO USE INTERCOM                  TYPE   1
    HOW TO USE THE TERMINAL              TYPE   2
    AN INTERACTIVE COMMAND               TYPE   3
    AN EDITOR COMMAND                    TYPE   4
    A REMOTE BATCH COMMAND               TYPE   5
```

As you can see, the display includes further instruction for using TEACH itself, as well as INTERCOM use.

Call TEACH if you need help in using commands.

TEACH FORMAT

To use TEACH, enter the command:

```
    TEACH
```

After the main display of TEACH appears, select an item for examination by typing the number of the item desired and pressing the RETURN key. The first page of the item chosen then appears.

At the end of each page displayed, this line appears:

```
    TO CONTINUE TYPE- GO.      TO END TYPE-   END.
```

These periods are sentence punctuators and are not required

The carriage remains at the end of the line. If you type GO, the next page of the current item is displayed. If the current item has no continuation, the main display of TEACH reappears.

To terminate the TEACH utility from the midst of an item sequence; type END. If the main display reappears, type END to exit from TEACH itself. END terminates TEACH only when the main display is waiting for you to type an item number or END.

If input response to the main TEACH display is not an item number or the word END, an error diagnostic appears with a request for a correct response. The carriage remains at the right.

```
       TEACH
     IF YOU WOULD LIKE TO KNOW ABOUT THE FOLLOWING,
   TYPE THE CORRESPONDING NUMBER, ELSE TYPE  END

   HOW TO USE INTERCOM                TYPE   1
   HOW TO USE THE TERMINAL            TYPE   2
   AN INTERACTIVE COMMAND             TYPE   3
   AN EDITOR COMMAND                  TYPE   4
   A REMOTE BATCH COMMAND             TYPE   5
   5


   INVALID REQUEST - TYPE OPTION NUMBER4
                                         ↑
                                         carriage remains here
                                         for correct entry
```

If the display is waiting for a user GO or END, an invalid type-in gives the message:

```
INVALID REQUEST.     TYPE GO OR END
```

A full entry of GO or END is required. Abbreviations cannot be used during TEACH operation.

/oldtext/=/newtext/ COMMAND

This EDITOR command allows a string of 1–20 characters in the edit file to be replaced by another character string. The strings may be the same or different lengths.

WHEN IS /oldtext/=/newtext/ USED?

This command is useful for changing the content of a line, as well as correcting errors. For example:

Only part of a line is in error. The incorrect characters can be changed without re-entering the whole line.

A correct line begins in the wrong column. By increasing or decreasing blanks before a word, the word position can be changed.

The same change is to be made in several lines. EDITOR locates and changes all occurrences.

The text to be replaced must be contained within a single line of the edit file. As many as 20 characters can be specified in either the original text or new text.

The tab character used in CREATE or ADD line entries is not recognized as a tab character with this command. The tab character is simply another character in a replacement string, it does not change a line position indicator.

At the end of command execution, the system specifies the number of changes made:

```
n CHANGES
```

If the message is 0 CHANGES when you expected at least 1, check that you have specified the proper line range, column number, and text string.

/oldtext/=/newtext/ FORMAT

The minimum parameters for the command that replaces a character string are:

```
/oldtext/=/newtext/,linenum
```

oldtext	String of 1–20 characters to be replaced
newtext	String of 0–20 characters to replace an existing string
linenum	Line number, or range of line numbers, to search for oldtext. It may have any of the following formats: ALL line number line-1, line-2 line-1, LAST

The delimiter must not be part of the string. Any characters are allowed.

Optional parameters can restrict the search for the /oldtext/ character string to a column position or a UNIT string, as they can with LIST and DELETE.

VETO OPTION OF /oldtext/=/newtext/

The VETO parameter of the text replacement command defers execution of the command until the user examines the projected result and authorizes command completion. It is used in the same way as the VETO parameter of the DELETE or SAVE commands of EDITOR.

VETO is initiated by EDITOR, in the absence of a user parameter, whenever the text search or replacement string exceeds the system limit of 20 characters.

Format of the option is simply the parameter name. It can appear anywhere after the replacement text.

VETO	Abbreviated V

When VETO is selected:

1. EDITOR finds a line with characters to be replaced and displays the line that would result from execution.

2. User examines the line and accepts or rejects the change with a YES or NO type-in.

3. EDITOR continues by searching for next line with characters to be replaced..

The user controls acceptance of any line displayed by entering one of the following:

YES	Change line as displayed.	Abbreviated Y
NO	Do not change displayed line. Actually, any character other than Y or C is acceptable in place of NO.	Abbreviated N
CONTINUE	Change this line as displayed, then continue with any subsequent change without displaying lines.	Abbreviated C

The VETO option is helpful to beginners in several ways. You can use VETO to verify that text replacement produces the results expected. More important, however, using VETO helps you avoid common errors in the text strings entered.

Even after you have gained confidence in using the text replacement command, continue VETO selection any time file contents are not known completely or when all occurrences of a text search string are not to be changed.

VETO combined with the UNIT parameter can be a powerful tool in correcting errors in a program. For example, assume that you had intended using a MOVE CORR statement to reformat your input data. Consequently, records INFILE and FORMATFILE have the same field names. Later you decide not to use the CORR option and want to change the INFILE field names to make them unique. Alternate ways of doing this include:

Enter a separate command for each line to be changed. This requires knowing the line number of each statement that must be changed.

Enter one command to change all occurrences of the field name using the UNIT and VETO options.

*EOR AND *EOF

*EOR and *EOF are special EDITOR entries. Their use during edit file creation provides special indicators that can be interpreted by other system routines as operating system logical records or file delimiters. All files have an ending terminator. Only for particular circumstances would the user need to indicate a file end, since INTERCOM supplies it for the user. These two symbols are useful, however, since they increase program options and, in the case of permanent files, can conserve system resources.

Beginning programmers often find the words record and file confusing. Consistency exists within routines internal to INTERCOM and the operating system, but this consistency is not always apparent to a user. What the user thinks of as a file of individual records appears to part of the operating system as a single record in a file and to other parts as a file. The issue is further complicated by record and file boundaries that can be interpreted by an internal routine when a user is not aware that the boundary existed in the first place.

The best advice for handling this situation is simply to accept apparent inconsistencies: for any given situation. learn the entry you must make to achieve the desired results. Do not be disturbed that you must enter an *EOR line to execute the AT END imperative on a file named INPUT but an *EOF line to execute it on a file with names such as INFILE. For now, use the specific character string in the circumstances listed below. *EOR and *EOF are applicable only as entries into the edit file.

WHEN ARE *EOR AND *EOF USED?

The *EOR statement is used:

To execute the AT END imperative for a file named INPUT.

To separate system logical records in a file prepared for BATCH execution. *EOR is the equivalent of a punch card with 7/8/9 multipunched in column 1.

EDITOR writes the equivalent of *EOR (7/8/9) at the end of any file referenced in a SAVE command.

The *EOF statement is used:

To execute the AT END imperative for files with names other than INPUT.

To indicate the end of a file for reference by a COPY routine.

*EOF is the equivalent of a punch card with 6/7/8/9 multipunched in column 1. INTERCOM writes an end-of-file indicator as appropriate for other INTERCOM commands.

When the user enters a SAVE command with MERGE parameter, EDITOR writes an end-of-record indicator to the local file. By returning the saved file to the edit file and listing it, you can see this indicator causes an *EOR line to be created, with a line number assigned.

```
..SAVE,SQUEEZE,MERGE,200
..SAVE,SQUEEZE,MERGE,300
..EDIT,SQUEEZE
..LIST,ALL
  200=THIS IS SAVED LINE 200
  201=*EOR
  300=THIS IS SAVED LINE 300
..
```

The end-of-record and end-of-file indicator exist after the second merge, but they are used internally and are not displayed with the edit file.

It is important to realize that the character string *EOR or *EOF exists only in the temporary edit file. When EDITOR makes the file local, or copies it for submission to a compiler, the terminators are translated to a format used internally by system routines. The characters * E O and R exists only in the edit file.

A test in a program for a character string *EOR can never be successful. The function that tests for end of data read by a COBOL program is the AT END imperative.

On the other hand, an EDITOR command to find and delete the character string *EOR and *EOF is possible. In the 3 lines listed previously, line 201 can be deleted by:

```
. . DELETE,ALL,/*EOR/
```

Of course, the *EOR can be changed to *EOF by the following:

```
. . /*EOR/=/*EOF/,ALL
```

Look at the example which follows to see how INTERCOM and the operating system use *EOR and *EOF and how you may use them to your advantage. This example is pertinent for conserving entries in the permanent files catalog the operating system maintains internally to keep track of permanent files. Each STORE command results in an entry. Table entries are at a premium, since a full catalog prevents new permanent files. A message P.F. DIRECTORY FULL, FILE NOT PERMANENT is returned to the terminal when STORE cannot complete.

Assume two local files exist. A third file is to be created, and all three files made permanent with the name COMB. As a result of the COPY, the system adds file terminators:

```
..FILES
--LOCAL FILES--
   PONE       PTWO
..COPY,PONE,TEMP
..COPY,PTWO,TEMP
..EDIT,TEMP,SEQ
..LIST,ALL,SUP
THIS IS CONTENTS OF PONE
*EOR
*EOF
THIS IS CONTENTS OF PTWO
..ADD,SUP
ENTER LINES
*EOF
THIS IS CONTENTS OF ADDED THIRD PROGRAM
=
..SAVE,COMB,NOSEQ
..STORE,COMB,MINE
..FILES
--LOCAL FILES--
   *COMB       TEMP        PONE        PTWO
..
```

You must enter the *EOF after ADD is initiated to delimit PTWO information from new lines entered; it is not supplied by the ADD command.

When you access the combined file at another terminal session, you can separate the programs or use them together. In order to move the third file to the edit file, use the operating system control statement COPYBF to copy the first two files to a scratch file, then copy the third file and move it to the edit file. At the end of the combined file copy, the system displays an informative message.

```
..FETCH,COMB,MINE
..COPYBF,COMB,P1AND2,2
..COPYBF,COMB,P3
 EOF/EOI ENCOUNTERED
..FILES
--LOCAL FILES--
   P3     *COMB      P1AND2
..EDIT,P3,SEQ
..LIST,ALL,SUP
THIS IS CONTENTS OF ADDED THIRD PROGRAM
..
```

If the contents of the first 2 programs are not needed during the current terminal session, an alternate way to select program 3 is:

```
..EDIT,COMB,SEQ
..LIST,ALL
   100=THIS IS CONTENTS OF PONE
   110=*EOR
   120=*EOF
   130=THIS IS CONTENTS OF PTWO
   140=*EOR
   150=*EOF
   160=THIS IS CONTENTS OF ADDED THIRD PROGRAM
..DELETE,100,150
..LIST,ALL
   160=THIS IS CONTENTS OF ADDED THIRD PROGRAM
..
```

Alternatively, LIST,ALL,/*EOF/ could have been entered to display the lines containing the *EOF. The third program begins on the line following the last *EOF.

COBOL CHARACTER SET AND COLLATING SEQUENCE A

CONTROL DATA operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE 1, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table are applicable to BCD terminals: ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

NOTE

In the following chart, characters identified by the heading CDC GRAPHIC are applicable to BCD-CRTs models: 214-11, 214-12, 217-11, and 217-12.

Characters identified by the heading ASCII GRAPHIC are applicable to ASCII (CRTs and TTYs) as follows:

ASCII-CRTs

217-13, 217-14, 731-12, 732-12

711-10

733-10

ASCII-TTYs

Model 33 or 35 Teletype

713-10

STANDARD CHARACTER SETS

CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code	CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code
:†	:	00††	8-2	00	8-2	072	6	6	41	6	06	6	066
A	A	01	12-1	61	12-1	101	7	7	42	7	07	7	067
B	B	02	12-2	62	12-2	102	8	8	43	8	10	8	070
C	C	03	12-3	63	12-3	103	9	9	44	9	11	9	071
D	D	04	12-4	64	12-4	104	+	+	45	12	60	12-8-6	053
E	E	05	12-5	65	12-5	105	–	–	46	11	40	11	055
F	F	06	12-6	66	12-6	106	*	*	47	11-8-4	54	11-8-4	052
G	G	07	12-7	67	12-7	107	/	/	50	0-1	21	0-1	057
H	H	10	12-8	70	12-8	110	(	(	51	0-8-4	34	12-8-5	050
I	I	11	12-9	71	12-9	111	)	)	52	12-8-4	74	11-8-5	051
J	J	12	11-1	41	11-1	112	$	$	53	11-8-3	53	11-8-3	044
K	K	13	11-2	42	11-2	113	=	=	54	8-3	13	8-6	075
L	L	14	11-3	43	11-3	114	blank	blank	55	no punch	20	no punch	040
M	M	15	11-4	44	11-4	115	, (comma)	, (comma)	56	0-8-3	33	0-8-3	054
N	N	16	11-5	45	11-5	116	. (period)	. (period)	57	12-8-3	73	12-8-3	056
O	O	17	11-6	46	11-6	117	≡	#	60	0-8-6	36	8-3	043
P	P	20	11-7	47	11-7	120	[	[	61	8-7	17	12-8-2	133
Q	Q	21	11-8	50	11-8	121	]	]	62	0-8-2	32	11-8-2	135
R	R	22	11-9	51	11-9	122	%	%	63††	8-6	16	0-8-4	045
S	S	23	0-2	22	0-2	123	≠	" (quote)	64	8-4	14	8-7	042
T	T	24	0-3	23	0-3	124	→	_ (underline)	65	0-8-5	35	0-8-5	137
U	U	25	0-4	24	0-4	125	∨	!	66	11-0 or	52	12-8-7 or	041
V	V	26	0-5	25	0-5	126				11-8-2†††		11-0†††	
W	W	27	0-6	26	0-6	127	∧	&	67	0-8-7	37	12	046
X	X	30	0-7	27	0-7	130	↑	' (apostrophe)	70	11-8-5	55	8-5	047
Y	Y	31	0-8	30	0-8	131	↓	?	71	11-8-6	56	0-8-7	077
Z	Z	32	0-9	31	0-9	132	<	<	72	12-0 or	72	12-8-4 or	074
0	0	33	0	12	0	060				12-8-2†††		12-0†††	
1	1	34	1	01	1	061	>	>	73	11-8-7	57	0-8-6	076
2	2	35	2	02	2	062	≤	@	74	8-5	15	8-4	100
3	3	36	3	03	3	063	≥	\	75	12-8-5	75	0-8-2	134
4	4	37	4	04	4	064	¬	^(circumflex)	76	12-8-6	76	11-8-7	136
5	5	40	5	05	5	065	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	073

†Twelve or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations from ASCII/EBCDIC % yield a blank (55_8).

†††The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

EDITOR COMMAND FORMAT SUMMARY B

```
ADD [,line [,incr]]  [,SUP]  [,OVERWRITE]

BYE [,BYE]

CREATE [,line [,incr]]  [,SUP]

          ( ALL                    )
DELETE,  <  line-1  [,  { line-2 } ] >  [,/text/ [,(col-1 [,col-2])] [,UNIT]]  [,VETO]
          ( LAST        { LAST   }  )

EDIT,filename [,SEQUENCE]

         [( ,format-name                                                  )]
FORMAT   [< [,TAB=c]  [,tab-1 [,tab-2 [ , . . . [,tab-n] ] ] ]  [,CH=nnn] >]
         [( ,SHOW                                                         )]

       [  ( ALL                      )]
LIST   [ , < line-1  [,  { line-2 } ] >]  [,SUP]  [,/text/ [,(col-1 [,col-2])]  [,UNIT]]
       [  ( LAST         { LAST   }   )]

RESEQ [,line[,incr]]

RUN,system-name [,FILE=filename] [,NOEX] [,SUP]

                                                [  ( ALL                      )]
SAVE,filename [,NOSEQ] [,OVERWRITE][,MERGE]     [ , < line-1  [,  { line-2 } ] >]
                                                [  ( LAST         { LAST   }   )]

        [,/text/ [,(col-1 [,col-2])] [,UNIT]]  [,VETO]

[ = ] linenum=text

                       [  ( ALL                      )]
/oldtext/=/newtext/    [ , < line-1  [,  { line-2 } ] >]  [,(col-1 [,col-2])]  [,UNIT]  [,VETO]
                       [  ( LAST         { LAST   }   )]
```

Legend

{ }	Signify that one of enclosed items may be selected
[ ]	Enclose optional parameters
—	Signify minimum abbreviation

GLOSSARY C

Attach

Execution of a FETCH or ATTACH command to make a permanent file available for use at the terminal.

Batch Processing

Execution by submission of a series of statements in a job deck, or deck image, starting with a job statement and ending with an end-of-information indicator. Contrast with execution through INTERCOM where a single statement is executed independently of any other statement.

Central Site

The location of the CDC CYBER 170, CYBER 70 or 6000 Series computer running INTERCOM and user programs. It may be many miles away or in another room in the same building. Equipment attached to the system includes card readers, line printers, magnetic tape units, and mass storage, and communication equipment for remote terminals.

Command

The form of an instruction to INTERCOM. It is terminated by pressing the RETURN key.

Command Mode

The initial state of INTERCOM after LOGIN. The word COMMAND appears to the left of a line when INTERCOM can accept another command. Contrast with editor mode.

Connected File

A file equated with the terminal. Input file data must be entered from the keyboard; output file data is displayed immediately. No copy of the file data remains in the system.

Edit File

Temporary work area that holds files being created or updated through the EDITOR utility of INTERCOM. Contrast with local file and permanent file.

Editor Mode

The state of INTERCOM existing after the EDITOR utility has been called. A file can be created or modified in the edit file work area. Contrast with command mode.

Interactive Processing

Utilization of connected files during program execution. The program must expect keyboard input; execution is supended until input is entered. Output is displayed immediately.

Local File

A file available for reference in INTERCOM commands. Appears under category LOCAL FILES when FILES command is executed.

Attached permanent files, connected files, and files referenced in the SAVE command are local files. A file becomes a local as a result of the LOCAL desposition of the BATCH command. Local files are destroyed by the LOGOUT command, unless they are permanent files. Contrast with permanent file and edit file.

Operating System Command

A command that is passed to other portions of the operating system for execution instead of execution by INTERCOM.

Permanent File

A file that exists from one terminal session to another. It is stored at the central site on mass storage. When called to the terminal in a subsequent session, it is said to be an attached permanent file. Contrast with local file and edit file.

Remote Terminal

A teletypewriter or display terminal connected to the central site computer through INTERCOM use of communication equipment.

Terminal Session

The period between execution of a LOGIN and LOGOUT command.

INDEX

COMMENT SHEET

MANUAL TITLE CDC INTERCOM Version 4 Interactive Guide for Users of COBOL

PUBLICATION NO. 60495100 REVISION B

FROM: NAME:

BUSINESS ADDRESS:

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

FOLD ON DOTTED LINES AND STAPLE

CUT ALONG LINE

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