SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

General Editor (EDT)

User Guide/Programmer Reference

UP-8828 Rev. 2

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) General Editor (EDT) User Guide/Programmer Reference", UP-8828 Rev. 2.

This revision documents the following new features for the 8.0 release:

- screen commands;
- screen mode processing capability for source code and data input;
- error file processing for correcting source code errors;
- EDT support for the COBOL editor;
- parameters on the @SET directive to specify the length of data your screen accepts or displays;
- HELP screen formats for all the EDT commands.

All other changes are corrections or expanded descriptions applicable to features present in EDT prior to the 8.0 release.

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8828 Rev. 1 and UP-8828 Rev. 1-A will be available for 6 months after the release of 8.0. Should you need additional copies of this edition, you should order them within 90 days of the release of 8.0. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00,A19,B00,B19,18,18U, 19,19U,20,20U,21,21U,28U,29U,75,75U, 76 and 76U (Cover and 268 pages)	Library Memo for UP-8828 Rev. 2
		RELEASE DATE:
		October, 1982

General Editor (EDT)

OS/3

User Guide/
Programmer Reference

H

SPERRY UNIVAC

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: UP-8828 Rev. 2
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover/Disclaimer								
PSS	1							
Preface	1 thru 4							
Contents	1 thru 7							
1	1 thru 9							
2	1 thru 4 4a 5 thru 8							
3	1 thru 75							
4	1 thru 10							
5	1 thru 13							
6	1 thru 8							
7	1 thru 6 6a 7 thru 16							
8	1 thru 12 12a 13 thru 15							
9	1, 2							
10	1 thru 26							
11	1 thru 17							
12	1							
Appendix A	1							
Appendix B	1, 2							
Appendix C	1							
Appendix D	1 thru 3							
Appendix E	1							
Appendix F	1 thru 11							
Appendix G	1 thru 21							
Index	1 thru 11							
User Comment Sheet								

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY UNIVAC Operating System/3 (OS/3). Specifically, this manual describes the OS/3 General Editor (EDT) and its effective use. Its intended audience is the novice user who has little knowledge in data processing, or the more experienced user who may or may not be familiar with EDT.

This manual is divided into the following sections:

- Section 1. EDT Concepts

  Introduces you to the functions of EDT.

- Section 2. Using EDT

  Describes how to access, use, and terminate EDT.

- Section 3. EDT Commands

  Explains the purpose of the EDT commands and how to use them.

- Section 4. Command Modifiers

  Explains the purpose of the command modifiers and how to use them.

- Section 5. EDT Procedure Files

  Explains the purpose and use of the EDT procedure files and the procedure file commands.

- Section 6. EDT Variables and Commands

  Explains the purpose of the EDT variables and how to define and use them.

- Section 7. Directives

  Explains the purpose of the directives and how to use them.

- Section 8. Screen Commands

  Explains the purpose of the screen commands and how to use them.

- Section 9. Function Key Usage

  Explains the purpose and use of the function keys available to you in EDT.

- Section 10. Screen Mode Processing

  Explains the purpose and use of EDT screen mode and gives a sample session.

- Section 11. Error File Processing

  Explains the purpose and use of the error file processor and gives a sample session.

- Section 12. Error Detection and Recovery

  Describes the types of errors that may occur during an EDT session.

- Appendixes

  Lists the minimum abbreviations for the commands, command modifiers, directives, procedure file commands, and keywords. Also, summarizes the EDT commands, describes the EDT processing loop, explains the differences between EDT and Basic Editor Monitor Editor (BEM EDT), and provides sample EDT sessions.

Other current OS/3 publications referenced in this manual that are helpful when using EDT are:

System 80 Environment

- COBOL editor user guide/programmer reference, UP-9106.

  Describes the formats used to create COBOL source programs.

- Consolidated data management concepts and facilities, UP-8825

  Describes the organization and record formats of various file types.

- Interactive services commands and facilities user guide/programmer reference, UP-8845

  Describes the commands and operating procedures for workstation terminals.

- Job control user guide, UP-8065

  Describes the job control language used under OS/3.

- RPG II editor user guide/programmer reference, UP-8803

  Describes the formats used to create RPG II source programs.

- System service programs (SSP) user guide, UP-8841

  Describes various system utilities (e.g., librarian, linkage editor).

- System messages programmer/operator reference, UP-8076

  Describes the errors encountered when using OS/3.

- Spooling and job accounting concepts and facilities, UP-8869

  Describes basic spooling and job accounting concepts and options available to control spooling systems.

Series 90 Environment

- COBOL editor user guide/prcgrammer reference, UP-9106

  Describes the formats used to create COBOL source programs.

- Consolidated data management concepts and facilities, UP-8825

  Describes the organization and record formats of various file types.

- Interactive services commands and facilities user guide/programmer reference, UP-8845

  Describes the commands and operating procedures for the workstation terminals.

- Job control user guide, UP-8065

  Describes the job control language used under OS/3.

- RPG II editor user guide/programmer reference, UP-8803

  Describes the formats used to create RPG II source programs.

- System service programs (SSP) user guide, UP-8062

  Describes various system utilities (e.g., librarian, linkage editor).

- System messages programmer/operator reference, UP-8076

  Describes the errors encountered when using OS/3.

- Spooling and job accounting concepts and facilities, UP-8869

  Describes basic spooling and job accounting concepts and options available to control spooling systems.

Contents

APPENDIXES

F. COMMAND SUMMARY

G. SAMPLE EDT SESSIONS

INDEX

USER COMMENT SHEET

FIGURES

TABLES

1. Introduction

1.1. WHAT IS EDT?

The SPERRY UNIVAC OS/3 General Editor, commonly known as EDT, is a user-oriented interactive program that enables you to create and update library modules and data files from a workstation. With EDT, you can interactively create and update your source programs, job control streams, and data files. You can also copy and concatenate files.

EDT supports three subprocessors: a COBOL subeditor, an RPG II subeditor, and an error file processor.

The language subeditors allow you to create and update COBOL or RPG II source programs interactively at your workstation using screens, and they provide immediate syntax checking. For more information on them, see either the COBOL editor user guide/programmer reference or the RPG II editor user guide/programmer reference.

The error file processor lets you see and correct errors in COBOL, RPG II, and FORTRAN IV source code immediately after compilation, without waiting for your language compiler to print its error listing. For more information on the error file processor, see Section 11 of this manual.

NOTE:

As part of its operation, EDT uses the screen format file $Y$FMT. Therefore, whenever you're using EDT, you cannot update $Y$FMT using the MIRAM librarian (MLIB) because MLIB needs exclusive use of $Y$FMT for that operation.

1.2. HOW EDT OPERATES

You create and update files (via EDT) in a temporary MIRAM disk file known as the EDT work-space file. The work-space file is automatically created each time EDT is activated and lasts for the duration of the EDT session.

When you create a file, you enter data or statements into the work-space file as shown in Figure 1-1. When all entries are made, you can then save a copy of the work-space file by writing it to a permanent file, listing it on the printer, or punching it on cards.

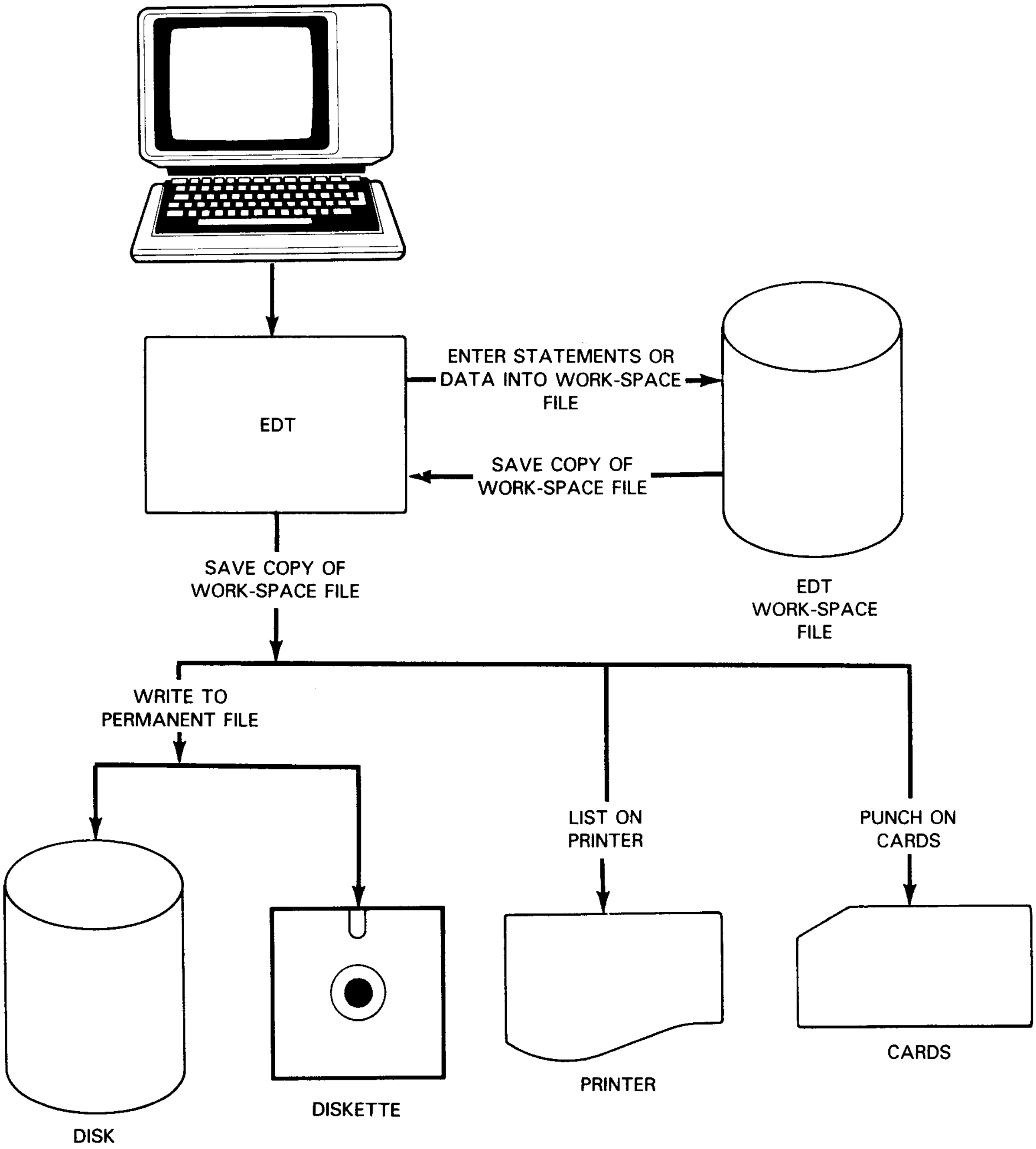

EDT
ENTER STATEMENTS OR DATA INTO WORK-SPACE FILE
SAVE COPY OF WORK-SPACE FILE
EDT WORK-SPACE FILE
SAVE COPY OF WORK-SPACE FILE
WRITE TO PERMANENT FILE
LIST ON PRINTER
PUNCH ON CARDS
DISK
DISKETTE
PRINTER
CARDS

Figure 1—1. Creating a Library Module or Data File via EDT

When you update an existing file, a copy of that file is placed in the work-space file (Figure 1-2). The original version of your file remains unchanged as a backup copy with all updating being done in the work-space file. Once you finish updating your work-space file, you can write a copy of it to the original file (overwriting the original version) or write it to a new file. You can also list it on the printer or punch it on cards.

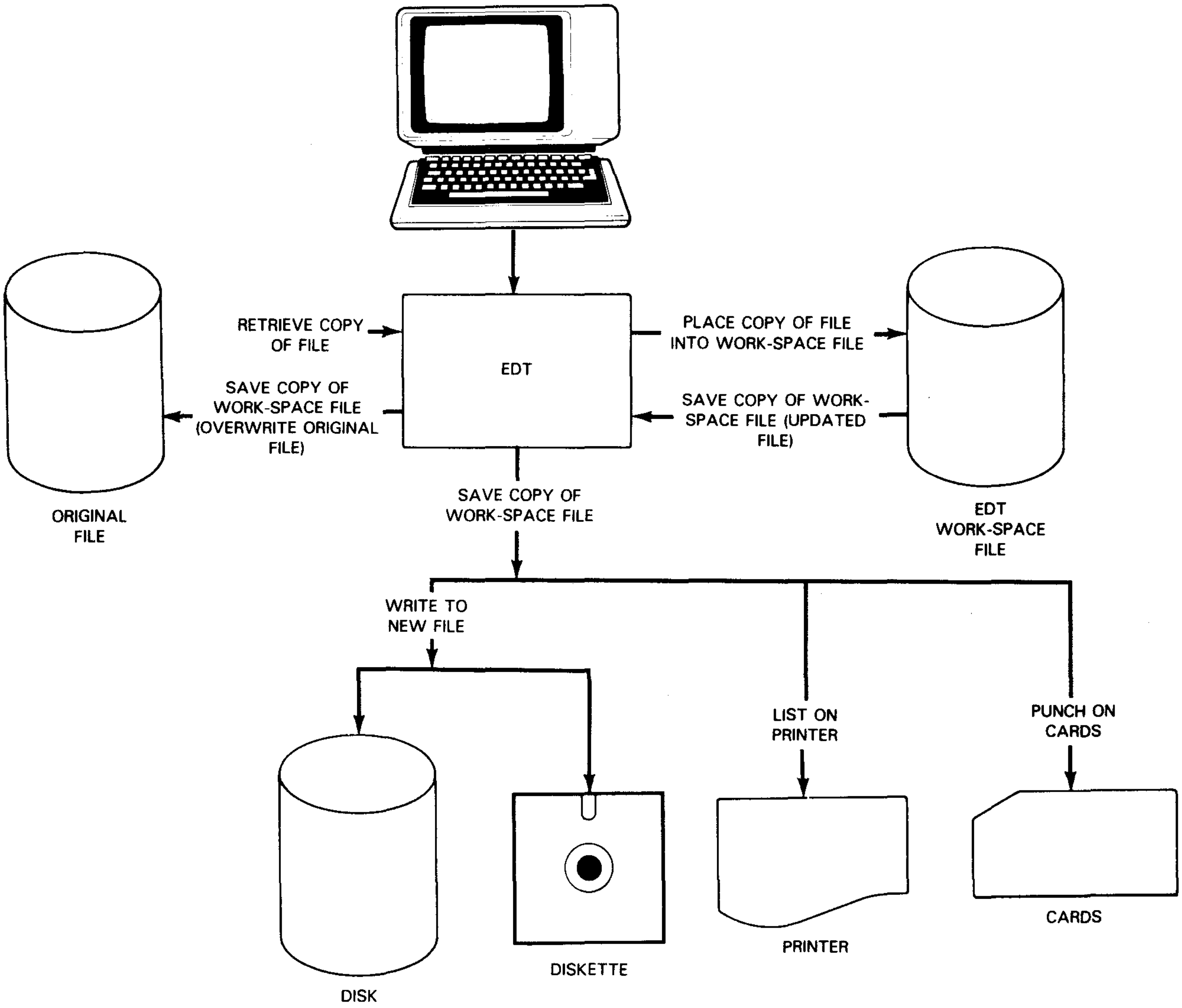

RETRIEVE COPY OF FILE
SAVE COPY OF WORK-SPACE FILE (OVERWRITE ORIGINAL FILE)
ORIGINAL FILE
EDT
PLACE COPY OF FILE INTO WORK-SPACE FILE
SAVE COPY OF WORK-SPACE FILE (UPDATED FILE)
EDT WORK-SPACE FILE
SAVE COPY OF WORK-SPACE FILE
WRITE TO NEW FILE
LIST ON PRINTER
PUNCH ON CARDS
DISK
DISKETTE
PRINTER
CARDS

Figure 1—2. Updating a Library Module or Data File via EDT

During an EDT session, EDT assigns temporary work-space line numbers to data or statements entered in the work-space file. Each entry made to the work-space file is assigned its own unique line number; by using these line numbers you can reference and manipulate the data and statements in your file. The line numbers are 8-digit numbers with leading zeros suppressed. For example, a number such as 0001.2345 would be displayed as 1.2345. Note that a decimal point is used in the line number for convenience. Work-space line numbers are automatically generated and displayed by EDT.

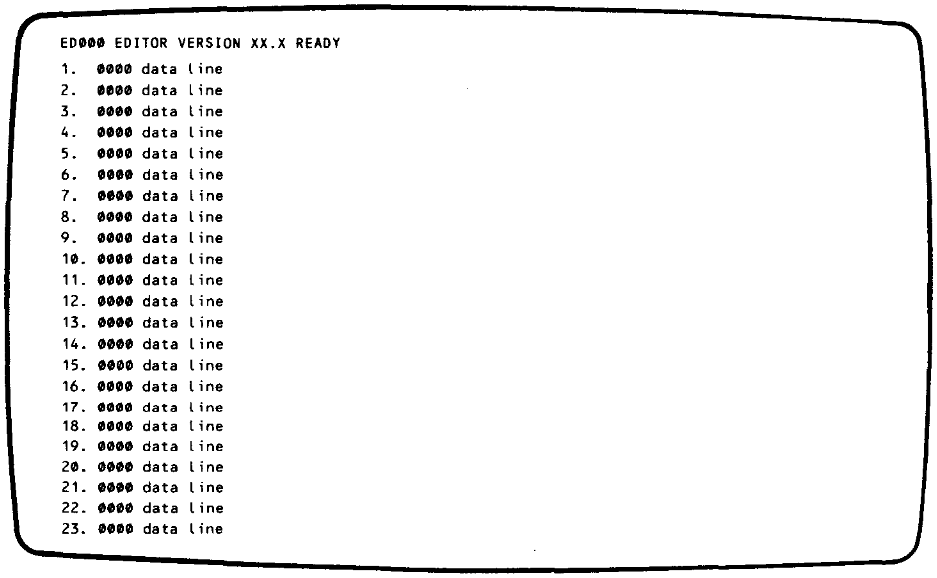
When using EDT, you can operate in one of two modes: line mode or screen mode. The following subsections describe both of these modes and help give you an idea of which mode is more suited to your applications. Unless you specify screen mode, EDT automatically operates in line mode. You specify screen mode by using the EDT directive @SET, which we describe in Section 7.

1.2.1. Line Mode

In line mode, EDT lets you enter source code or data one line at a time. After completing the line, you press the XMIT key to pass the line to the EDT work-space file. If you have only a few lines of source code to enter, line mode would be the quickest and easiest way to enter them. As you key in lines, you're unable to move the cursor back up to change those lines and you can see only 24 lines at a time on your screen.

In line mode, you can use all the EDT commands. If you're operating in line mode when you enter a command, EDT returns you to line mode after it processes the command.

The following is an example of a typical line mode screen:



```
ED000 EDITOR VERSION XX.X READY
1.  0000 data line
2.  0000 data line
3.  0000 data line
4.  0000 data line
5.  0000 data line
6.  0000 data line
7.  0000 data line
8.  0000 data line
9.  0000 data line
10. 0000 data line
11. 0000 data line
12. 0000 data line
13. 0000 data line
14. 0000 data line
15. 0000 data line
16. 0000 data line
17. 0000 data line
18. 0000 data line
19. 0000 data line
20. 0000 data line
21. 0000 data line
22. 0000 data line
23. 0000 data line
```

1.2.2. Screen Mode

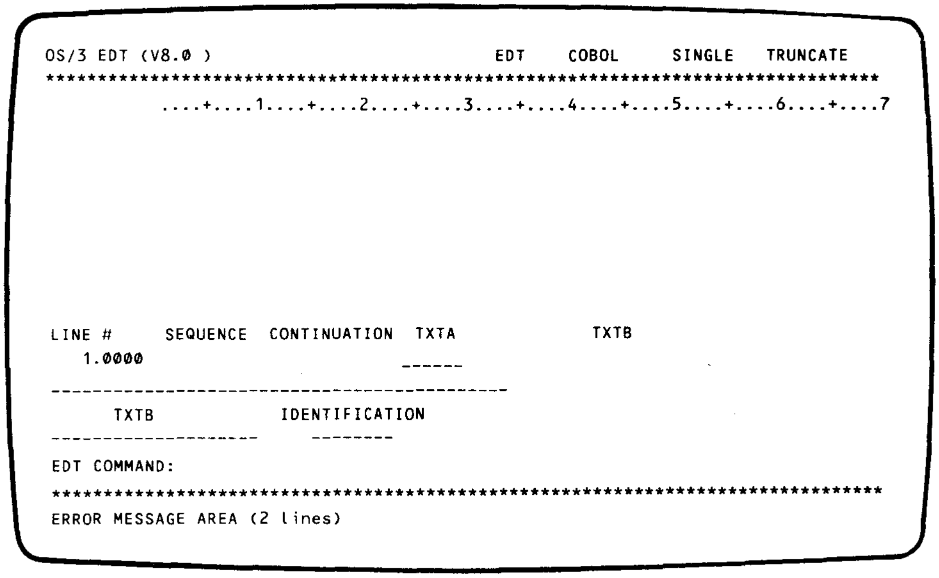
In screen mode, EDT lets you enter up to 14 lines of source code or data at one time. Again, you press the XMIT key to pass the lines to the EDT work space. However, depending on whether you want to enter single or multiple lines in one transmission, either : (1) EDT displays the single lines you key in as you transmit them to the EDT work space and scrolls the single lines up as it fills the screen; or (2) you enter up to 14 lines at a time before you transmit them. In these ways, EDT screen mode makes it easier for you to enter source code or data.

Another advantage to using screen mode is that EDT provides formatted screens for entering source code or data. In screen mode, you can enter source code using supplied COBOL, RPG II, or FORTRAN IV screens. These screens identify the programming fields. In addition, EDT provides a freeform screen that you can design to accept uniform data. For example, you can create a name and address file showing the appropriate fields or you can create an ASSEMBLER coding form.

If you are a COBOL or RPG II programmer, you may have used or considered using the EDT subeditors (the COBOL editor or the RPG II editor) for creating and updating your source code. If your updates are not extensive and you don't need the syntax checking provided by those editors, you can use screen mode to quickly make updates and changes.

You activate EDT screen mode using the @SET directive described in Section 7. For a complete description of screen mode processing, see Section 10. Once in screen mode, you can use all the EDT commands except the UPDATE command. If you're operating in screen mode when you enter a command, EDT returns you to screen mode after it processes the command.

The following is a sample screen-mode screen. This screen helps you enter COBOL source code:



```
OS/3 EDT (V8.Ø )                             EDT    COBOL     SINGLE   TRUNCATE
********************************************************************************
           ....+....1....+....2....+....3....+....4....+....5....+....6....+....7








LINE #     SEQUENCE  CONTINUATION  TXTA               TXTB
   1.ØØØØ                         ------
--------------------------------------------
      TXTB              IDENTIFICATION
--------------------    --------
EDT COMMAND:
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

1.3. COMMAND CONVENTIONS AND TERMS

This is a list of conventions that you must follow when formatting EDT commands:

- All commands must begin with the current command trigger, which is preset to an at sign (@):

  ```
  @PRINT
  ```

- All commands keyed in as data lines must begin with a double command trigger:

  ```
  @@DELETE
  ```

- Commands, command modifiers, and parameters in capital letters must be keyed in exactly as shown:

  ```
  @LIST
  ```

- Underscoring part of a command, command modifier, parameter, or keyword indicates that it may be abbreviated; you need to enter only the underscored portion. However, you may enter the entire command and it will execute properly. When a command, command modifier, parameter, or keyword is not underscored, you must enter the entire word. In the following example, PU of the PUNCH command is underscored. You must enter at least PU for the command to execute; but you may enter PUN, PUNC, or PUNCH and the command will execute properly.

  ```
  @PUNCH
  ```

- Parameters constructed in lowercase letters designate user-defined variables:

  ```
  @FIND 'search-string'
  ```

- Information contained within brackets represents optional entries that are included or omitted (depending upon command requirements):

  ```
  @NOP [comment]
  ```

- Alternate choices for a parameter, when at least one choice is required, are enclosed in braces:

  ```
  @GOTO {label}
        {line }
  ```

- Commas, periods, colons, semicolons, equal signs, and parentheses must be keyed in exactly as shown:

  ```
  @SET LINE=length
  ```

- A shaded parameter indicates that this parameter is the default:

  @CHECK $\left[\begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}\right]$

This is a list of terms that represent variable information you supply when structuring your commands:

- change-string

  Specifies a character string (1 to 50 characters) enclosed in apostrophes or a hexadecimal string enclosed in apostrophes and preceded by an X.

- column-range

  Specifies columns in the work-space file with column numbers, the special symbols [and], or EDT variables. Special symbols or EDT variables can be used only if they are assigned appropriate values. A single column, nonconsecutive columns, consecutive columns, or a combination of columns may be specified. Nonconsecutive columns are separated by commas (1,5,7) and consecutive columns are specified by the first and last columns separated by a colon (5:10). Columns must be specified in ascending order.

- destination

  Specifies by a line number or multiple line numbers the line location where lines are copied (via the COPY command) or moved (via the MOVE command).

- increment

  Defines a 1- to 9-digit decimal number used as the increment in the SEQUENCE and NUMBER commands.

- label

  Defines a symbolic name of a command line. A label can consist of one to seven alphanumeric characters, but the first must be alphabetic. A label can be defined in any EDT command or in any EDT procedure file command. It must be enclosed in parentheses and immediately follow the @ sign to be valid. It cannot contain embedded spaces. For example, (XYZ) is the label in this command line:

  ```
  7.0000 @(XYZ)  PRINT 10:20
  ```

  NOTE:

  When the label is specified in the GOTO command, parentheses should not be used: GOTO XYZ

- line-range

Specifies by work-space line numbers, special symbols % * $ & and ? (if they have appropriate values) and EDT variables (if they have appropriate values), lines in the work-space file. A single line, nonconsecutive lines, consecutive lines, or any combination of these lines may be specified. Nonconsecutive lines are separated by commas (3,8,11) and consecutive lines are specified by the first and last lines separated by a colon (*:30). Lines must be specified in ascending order.

- search-string

Defines a character string or a hexadecimal string that you want EDT to search for. The character string (of 1 to 50 characters) must be enclosed in apostrophes (') or quotes(''). The hexadecimal string must be enclosed in apostrophes and be preceded by an X. When apostrophes are used, any occurrence of the specified string is searched for. When quotes are used (at the beginning, end, or at both ends of the string), only those strings bound by delimiters where the quotes appear are searched for. EDT considers the following characters to be text string delimiters: . (+) ! ; - / , ? : = '' * ' or a space.

If the search string itself contains quotes or apostrophes, you must specify both the quotes or apostrophes in the string and the quotes or apostrophes that delimit the string. For example, if the search string is 'xyz', you must specify the search string as '''xyz''' so EDT can access it correctly.

The following examples explain the difference between the use of apostrophes and quotes with a search string.

'ABC'	Searches for any occurrence of ABC
"ABC'	Searches for occurrences of ABC with a delimiter in front of A: /ABC
'ABC"	Searches for occurrences of ABC with a delimiter immediately after the C: ABC?
"ABC"	Searches for occurrences of ABC with delimiters on both ends: *ABC*

- sequence-string

Specifies an alphanumeric character string that EDT uses for sequencing (or ordering). The sequence string must be enclosed in apostrophes and at least one of its rightmost characters must be an integer from 0 to 9. A sequence string of up to 15 characters may be specified. Only the numeric characters on the right end of the sequence string are incremented, and all alphabetic characters to the left of the rightmost alphabetic character (including that character) are not changed.

- string

  Specifies a character string (of 1 to 50 characters) enclosed in apostrophes or a hexadecimal string enclosed in apostrophes preceded by an X.

- special symbols

  Defines characters that represent a present line range or column range in the work-space file.

  % Represents the line number of the lowest line in the current work-space file.

  $ Represents the line number of the highest line in the current work-space file.

  * Represents the line number of the current line in the work-space file.

  & Represents all lines in the work-space file (i.e., the entire work-space file). The line range for & is preset to .0001:9999.

  ? Represents the line number of the first line containing the most recently specified search string. EDT assigns this special symbol when you do an @ FIND command.

  [Represents the column number where the most recently found search string begins. EDT assigns this special symbol when you do an @ FIND command.

  ] Represents the column number where the most recently found search string ends. EDT assigns this special symbol when you do an @ FIND command.

- command help screens

  EDT provides help screens for all the commands (meaning the EDT commands, modifiers, directives, procedure file commands, variables, and screen commands) presented in this manual. To see these help screens, you use the @PROMPT screen command described in Section 8. These help screens give a short description of each command, along with its format. In this manual, we present the format for each command, as well as the format that the help screen displays for that command if you request it.

  You should note that the formats presented in this manual and those you see on your screen differ slightly. These differences exist because the workstation screen (or one of the display devices that supports EDT) cannot display all the characters used in the formats in this manual. Although the syntax appears to be different, you should specify the EDT commands according to the conventions we present in the preceding paragraphs. For a list of the differences in conventions, see the explanation for the @PROMPT screen command in Section 8.

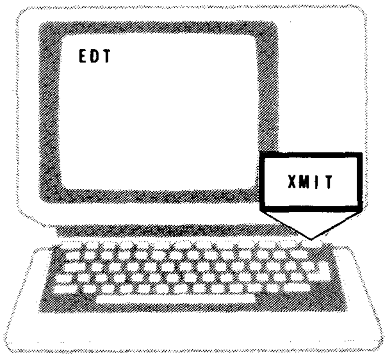
2. Using EDT

2.1. HOW TO ACCESS EDT

You can access EDT interactively* via a workstation, the System 80 console workstation, or a remote communications terminal that uses interactive services. When using the editor on a remote communications terminal that uses interactive services, be sure to use the field-protect feature on the terminal. Otherwise, you'll encounter errors.

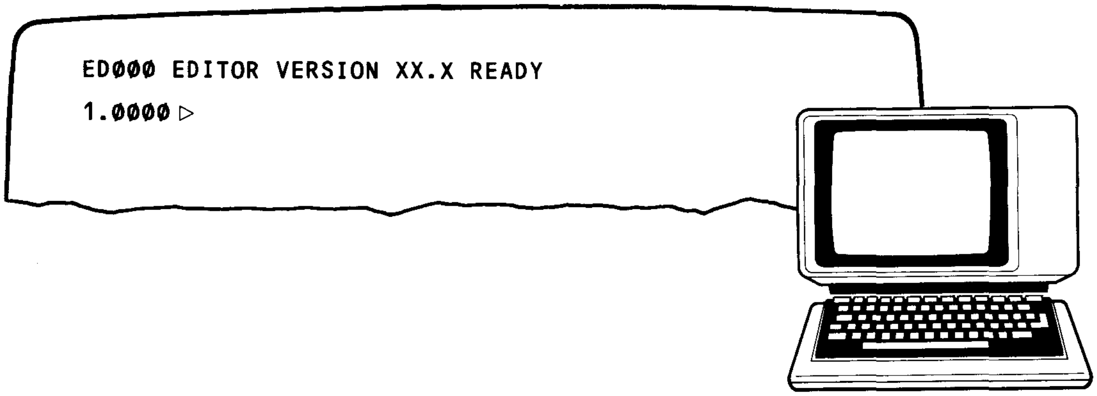
If you're accessing EDT via a workstation (and you're the first user), turn on the workstation and log on. (You can follow the logon procedure described in the interactive services commands and facilities user guide/programmer reference.) After you've successfully logged on, press the FUNCTION key and (while holding it down) press the SYS MODE key. Then key in EDT (and optionally the first input line to EDT) and transmit (press the XMIT key). EDT is now activated.

To access EDT via the System 80 console workstation, you press the FUNCTION key and WS MODE key at the same time and then transmit. Next, log on (again using the logon procedure described in the interactive services commands and facilities user guide/programmer reference). After you've successfully logged on, press the FUNCTION and SYS MODE keys at the same time. Then key in EDT (and optionally the first input line to EDT) and transmit. EDT is activated. When working at the console workstation, use EDT in workstation mode because the workstation screen is easier to use in that mode than in console mode.



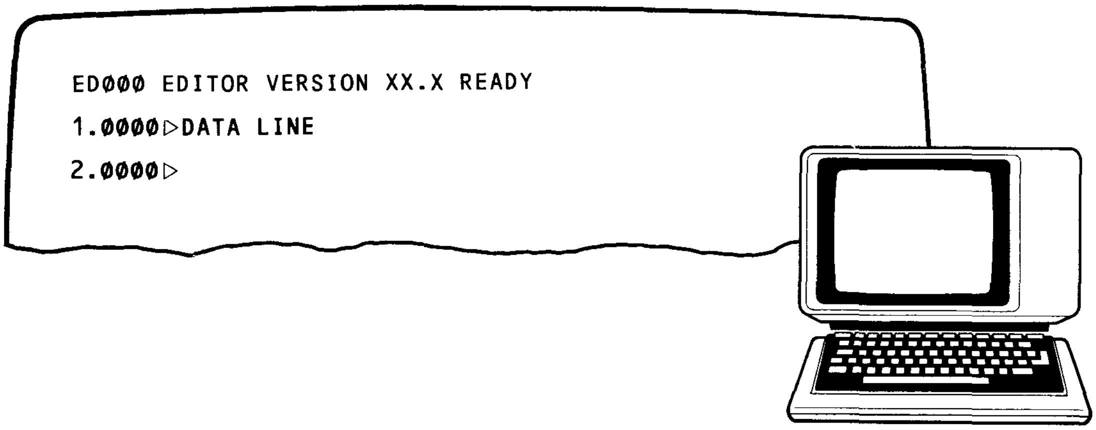
EDT can also be used in batch mode. See 2.4 for more details.

When EDT is successfully activated, it clears the workstation and displays the message EDITOR VERSION XX.X READY on the bottom of the screen. (The XX.X stands for the EDT version number that you're using. You can refer to it whenever discussing EDT with a Sperry Univac representative.) In addition, EDT sets the current work-space line to 1.0000 and displays the first line followed by a start-of-entry symbol (▷) on the next line of the screen. The work-space line number (0000.0000) and the start-of-entry symbol take up 10 columns on the screen but not in EDT's work space. So column 11 on the system state line equals column 1 of your data record.



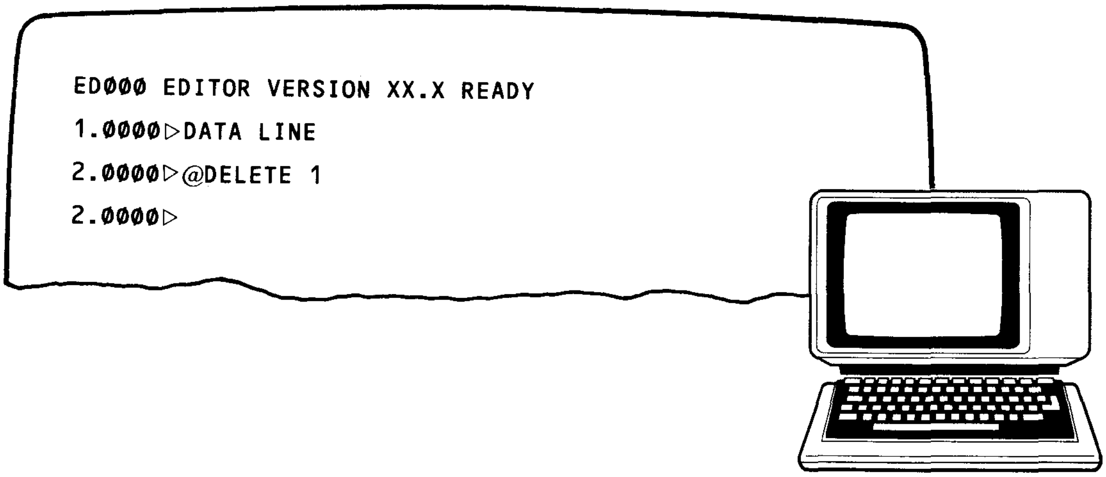
```
EDØØØ EDITOR VERSION XX.X READY
1.ØØØØ ▷
```

You may begin an EDT session by keying in either a data line (data or a statement) or an EDT command on line 1. After the line is keyed in, you must press the XMIT key to pass the line to the EDT work-space file. (This is true for all lines keyed in at the workstation during an EDT session.) If the line you key in is a data line (a line without a command trigger (@) or a line preceded by a double command trigger (@@)), the next line number displayed on the workstation screen is 2.0000. (At the beginning of an EDT session, the increment for the work-space line numbers is automatically set to 1.)



```
EDØØØ EDITOR VERSION XX.X READY
1.ØØØØ▷DATA LINE
2.ØØØØ▷
```

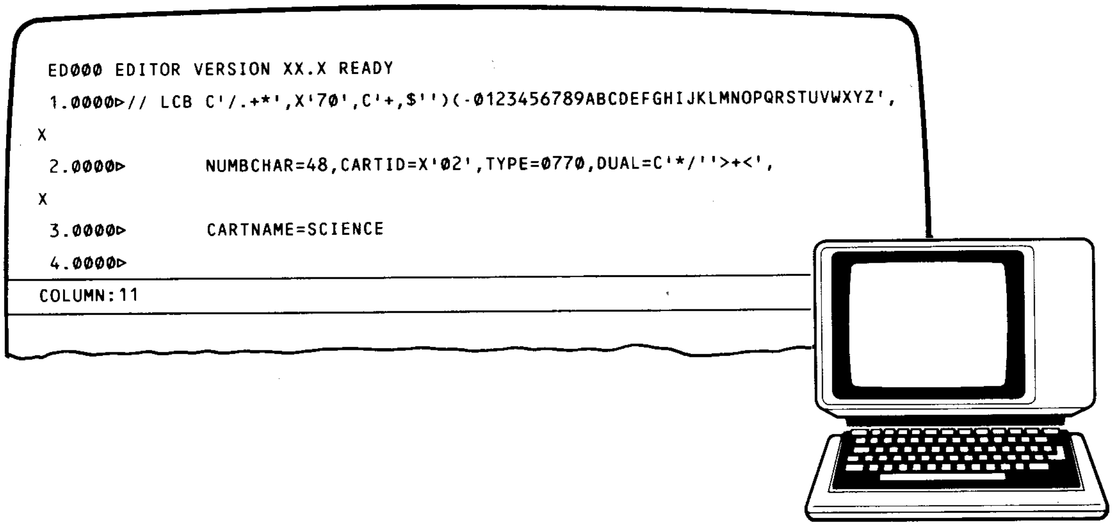
If the line you key in is a command preceded by a single command trigger (@), the next line number displayed is the same as the previous one. Therefore, if a command is keyed in on line 2.0000, the next line number displayed is 2.0000. There are; however, two exceptions to this: (1) if an @ command is entered to change the current line number (see 3.2), or (2) if a COPY or MOVE command is entered that transfers lines to new line locations beyond the highest line in the current work-space file. For example, if the highest line in the current work-space file is 75 and the command @MOVE 10 TO 100 is entered on line 2, the current work-space line number is automatically reset to 101, not 76. The destination (100) is determined by the increment, which is explicitly or implicitly specified in the destination (1), making the new line number 101. (See the COPY command (3.4) and MOVE command (3.10).)



```
ED∅∅∅ EDITOR VERSION XX.X READY
1.∅∅∅∅▷DATA LINE
2.∅∅∅∅▷@DELETE 1
2.∅∅∅∅▷
```

When entering EDT commands or data lines that are longer than one line (80 characters shown on the system state line) on the workstation screen, you do not have to indicate continuation at the end of the first line. EDT automatically moves the cursor to the beginning of the next line so you can enter the rest of your command or data line and transmit it to EDT's work space. However, some OS/3 products, such as job control, require an X in column 72 of the first line to show that a data record continues on the next line. In these cases, you must remember that column 11 on the system state line equals column 1 of your data record; so column 80 on the system state line equals only column 69 of your data record. Therefore, place the X for continuation in column 3 of the second line on EDT's screen. To get to column 3 of the second line, either press the space bar enough times to get the cursor there or set a tab at column 72 using the @SET directive (see Section 7). But don't transmit the line until you enter an X in column 3 of the second line. Then EDT will display the next line number to accept the rest of the data record.

For example, if you were keying in a // LCB job control statement that was too long to fit on one line, you would key it in on three consecutive lines, pressing XMIT after each one, like this:



```
 ED000 EDITOR VERSION XX.X READY
 1.0000▷// LCB C'/.+*',X'70',C'+,$'')(-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
X
 2.0000▷      NUMBCHAR=48,CARTID=X'02',TYPE=0770,DUAL=C'*/''>+<',
X
 3.0000▷      CARTNAME=SCIENCE
 4.0000▷
COLUMN:11
```

The rest of the EDT session continues in the same manner. A complete sample of an EDT session is provided in the following subsection.

During every EDT session, there are six types of commands you can enter. They are:

- EDT commands

  Let you create and update your files

- Modifier commands

  Enhance or restrict the performance of the EDT commands

- Procedure file commands

  Let you divide the EDT work-space file into a maximum of 10 subfiles

- Variable commands

  Hold values that give flexibility in structuring commands

- Directive commands

  Serve as the interface between you, EDT, and your operating system

- Screen commands

  Display screens that make EDT easier to use

Sections 3 through 8 describe each type of command in detail.

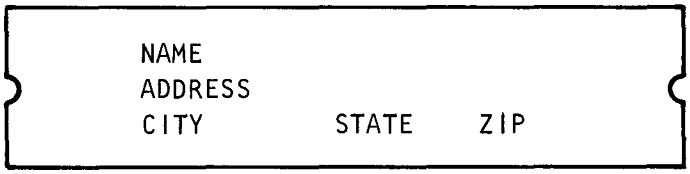
2.2. SAMPLE EDT LINE MODE SESSION

This sample EDT line mode session will give you a general idea of what an EDT session is and how it works. Since your knowledge of EDT may be limited, much of what is presented here may not be clear. It is only to serve as a base for what you will learn in the following sections.

Once you read through this manual it will be helpful to come back and examine this sample EDT session once again. It will then be clearer to you, and you can use it as a guide in your own EDT sessions.

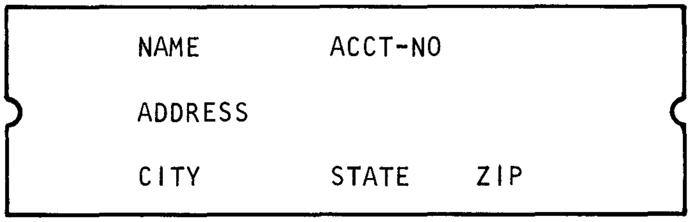
To see this same program updated in a screen mode session, see Appendix G, where we present more sample EDT sessions.

This EDT session is an update of an existing COBOL source program. The existing program prints address labels for magazines like this:



```
NAME
ADDRESS
CITY            STATE     ZIP
```

By using EDT, we will update that program to double-space the lines and include an account number next to the name like this:



```
NAME            ACCT-NO

ADDRESS

CITY            STATE     ZIP
```

We will begin this session by transferring a copy of the program LABELS into the EDT work-space file from the program library MAGAZINE on disk volume USER01. We will then update it, using the available EDT facilities. The session will end when we return a copy of the work-space file (the program LABELS) to the library. The old version of the program LABELS will be replaced by the updated version.

SAMPLE EDT LINE MODE SESSION

Transfers our source program into the work-space file

```
 1.ØØØØ▷@READ MO=LABELS,FIL=MAGAZINE,VSN=USERØ1
```

Displays the contents of the work-space file (our source program) on the workstation screen

```
51.ØØØØ▷@PRINT &
```

Our source program displayed

```
 1.ØØØØ▷      IDENTIFICATION DIVISION.
 2.ØØØØ▷      PROGRAM-ID.  LABELS.
 3.ØØØØ▷      ENVIRONMENT DIVISION.
 4.ØØØØ▷      CONFIGURATION SECTION.
 5.ØØØØ▷      SOURCE-COMPUTER.UNIVAC-OS3.
 6.ØØØØ▷      OBJECT-COMPUTER.UNIVAC-OS3.
 7.ØØØØ▷      INPUT-OUTPUT SECTION.
 8.ØØØØ▷      FILE CONTROL.
 9.ØØØØ▷      SELECT CARDIN, ASSIGN TO CARD READER-CARDIN-F.
10.ØØØØ▷      SELECT PRINTOUT, ASSIGN TO PRINTER-PRINTOUT-F.
11.ØØØØ▷      DATA DIVISION.
12.ØØØØ▷      FILE SECTION.
13.ØØØØ▷      FD  CARDIN
14.ØØØØ▷          LABEL RECORDS ARE OMITTED.
15.ØØØØ▷      Ø1  CARD-INPUT.
16.ØØØØ▷          Ø2  NAME        PIC X(25).
17.ØØØØ▷          Ø2  STREET      PIC X(25).
18.ØØØØ▷          Ø2  CITY        PIC X(15).
19.ØØØØ▷          Ø2  STATE       PIC X(2).
20.ØØØØ▷          Ø2  ZIP         PIC X(5).
21.ØØØØ▷          Ø2  FILLER      PIC X(8).
22.ØØØØ▷      FD  PRINTOUT
23.ØØØØ▷          LABEL RECORDS ARE STANDARD.
24.ØØØØ▷      Ø1  PRINTLINE       PIC X(29).
25.ØØØØ▷      WORKING-STORAGE SECTION.
26.ØØØØ▷      Ø1  CITY-STATE-ZIP-LINE.
27.ØØØØ▷          Ø2  CITY-OUT    PIC X(15).
28.ØØØØ▷          Ø2  FILLER      PIC X(1) VALUE SPACES.
29.ØØØØ▷          Ø2  STATE-OUT   PIC X(2).
30.ØØØØ▷          Ø2  FILLER      PIC X(2) VALUE SPACES.
31.ØØØØ▷          Ø2  ZIP-OUT     PIC X(5).
32.ØØØØ▷      PROCEDURE DIVISION.
33.ØØØØ▷      BEGIN-JOB.
34.ØØØØ▷          OPEN INPUT CARDIN, OUTPUT PRINTOUT.
35.ØØØØ▷      READ-CARD.
36.ØØØØ▷          READ CARDIN, AT END GO TO END-OF-JOB.
37.ØØØØ▷             MOVE SPACES TO PRINTLINE.
38.ØØØØ▷             WRITE PRINTLINE.
39.ØØØØ▷          MOVE NAME TO PRINTLINE.
40.ØØØØ▷          WRITE PRINTLINE.
41.ØØØØ▷          MOVE STREET TO PRINTLINE.
42.ØØØØ▷          WRITE PRINTLINE.
43.ØØØØ▷          MOVE CITY TO CITY-OUT.
44.ØØØØ▷          MOVE STATE TO STATE-OUT.
45.ØØØØ▷          MOVE ZIP TO ZIP-OUT.
46.ØØØØ▷          WRITE PRINTLINE FROM CITY-STATE-ZIP-LINE.
47.ØØØØ▷          GO TO READ-CARD.
48.ØØØØ▷      END-OF-JOB.
49.ØØØØ▷          CLOSE CARDIN, PRINTOUT.
50.ØØØØ▷          STOP RUN.
```

(continued)

```
New line to program keyed      51.0000▷          02  ACCT-NO      PIC X(4).
in at the workstation
Moves the contents of line     52.0000▷@MOVE 51 TO 16.5
51 to line 16.5
Changes the '8' on line 21     52.0000▷@ON 21  CHANGE '8' to '4'
to '4'
New line to program keyed      52.0000▷          MOVE ACCT-NO TO PRINTLINE.
in at the workstation
Moves the contents of line     53.0000▷@MOVE 52 TO 39.5
52 to line 39.5
New lines to program keyed     53.0000▷          MOVE SPACES TO PRINTLINE.
in at the workstation          54.0000▷          WRITE PRINTLINE.
Copies the contents of line    55.0000▷@COPY 53 TO 40.2, 42.2, 46.2
53 to lines 40.2, 42.2 and
46.2
Deletes line 53                55.0000▷@DELETE 53
Copies the contents of line    55.0000▷@COPY 54 TO 40.4, 42.4, 46.4
54 to lines 40.4, 42.4 and
46.4
Deletes line 54                55.0000▷@DELETE 54
Displays lines 12 through      55.0000▷@PRINT 12:31
31 of our work-space file
                               12.0000▷      FILE SECTION.
                               13.0000▷      FD  CARDIN
                               14.0000▷          LABEL RECORDS ARE OMITTED.
                               15.0000▷      01 CARD-INPUT.
                               16.0000▷         02 NAME         PIC X(25).
New line added                 16.5000▷         02 ACCT-NO      PIC X(4).
                               17.0000▷         02 STREET       PIC X(25).
                               18.0000▷         02 CITY         PIC X(15).
                               19.0000▷         02 STATE        PIC X(2).
                               20.0000▷         02 ZIP          PIC X(5).
Change made to existing line   21.0000▷         02 FILLER       PIC X(4).
                               22.0000▷      FD PRINTOUT
                               23.0000▷         LABEL RECORDS ARE STANDARD.
                               24.0000▷      01 PRINTLINE   PIC X(29).
                               25.0000▷      WORKING-STORAGE SECTION.
                               26.0000▷      01 CITY-STATE-ZIP-LINE.
                               27.0000▷         02 CITY-OUT     PIC X(15)..
                               28.0000▷         02 FILLER       PIC X(1) VALUE SPACES.
                               29.0000▷         02 STATE-OUT    PIC X(2).
                               30.0000▷         02 FILLER       PIC X(2) VALUE SPACES.
                               31.0000▷         02 ZIP-OUT      PIC X(5).
Displays lines 39-48           55.0000▷@PRINT 39:48
of work-space file             39.0000▷          MOVE NAME TO PRINTLINE.
New line added                 39.5000▷          MOVE ACCT-NO TO PRINTLINE.
                               40.0000▷          WRITE PRINTLINE.
New line added                 40.2000▷          MOVE SPACES TO PRINTLINE.
New line added                 40.4000▷          WRITE PRINTLINE.
                               41.0000▷          MOVE STREET TO PRINTLINE.
                               42.0000▷          WRITE PRINTLINE.
New line added                 42.2000▷          MOVE SPACES TO PRINTLINE.
New line added                 42.4000▷          WRITE PRINTLINE.
                               43.0000▷          MOVE CITY TO CITY-OUT.
                               44.0000▷          MOVE STATE TO STATE-OUT.
                               45.0000▷          MOVE ZIP TO ZIP-OUT.
```

(continued)

```
46.0000▷          WRITE PRINTLINE FROM CITY-STATE-ZIP-
                  LINE.
```

New line added

New line added

```
46.2000▷          MOVE SPACES TO PRINTLINE.
46.4000▷          WRITE PRINTLINE.
47.4000▷          GO TO READ-CARD.
48.0000▷   END-OF-JOB.
```

Lists the contents of the work-space file (our entire program)

```
55.0000▷@LIST &
```

Punches lines 12 through 45 on cards

```
55.0000▷@PUNCH 12:45
```

Writes our program to a permanent file on disk

```
55.0000▷@WRITE MO=LABELS,FIL=MAGAZINE,VSN=USER01
```

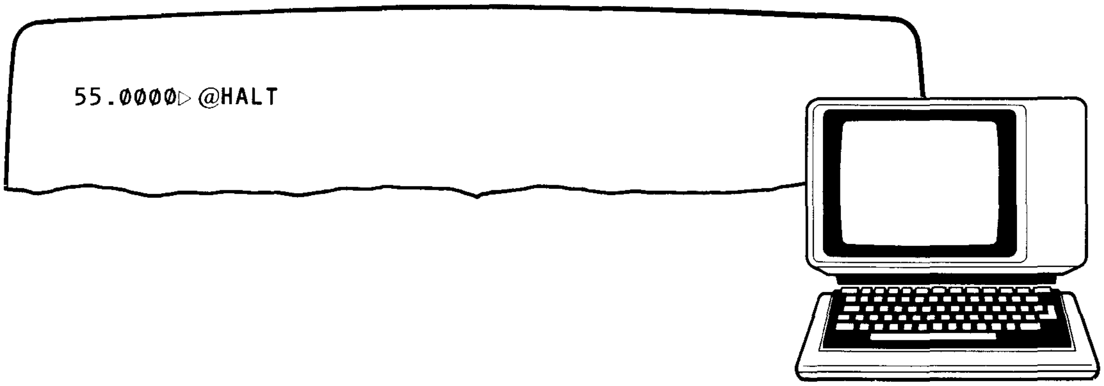
Reminds us that this module already exists, and asks if we want to overwrite it

We select Y, causing the previous version of our program to be overwritten

```
IS100 FILE/MODULE ALREADY EXISTS;
OK TO WRITE TO IT? (Y,N) Y
```

2.3. HOW TO TERMINATE EDT

To terminate EDT, key in @HALT and then transmit (press the XMIT key). This returns you to an idle workstation mode from which you can enter system mode to issue another interactive services command.



```
55.0000▷ @HALT
```

If you want to temporarily leave EDT (and return to system mode), key in @SYSTEM instead of @HALT. When you are ready to return to EDT, key in RESUME (in system mode).

2.4. USING EDT IN A BATCH ENVIRONMENT

Even though EDT is an interactive product, it is important to note that you can also run your EDT session as a batch utility. For example, if you wanted to make a change throughout your whole work-space to a source program using EDT but you didn't have a workstation immediately available, you could make the change by running the EDT session as a batch utility. From LOGON to LOGOFF you enter on cards exactly what you would key in at the workstation during an interactive EDT session (as in Figure 2-1). Note that you must include a // FILEID card in front of the card input and a // FIN card at the end of the card input when running EDT as a batch utility. Both the // FILEID card and // FIN card are described in the job control user guide.

NOTE:

You may run any previously stored EDT session as a batch utility by simply issuing the appropriate ENTER command from the system console.

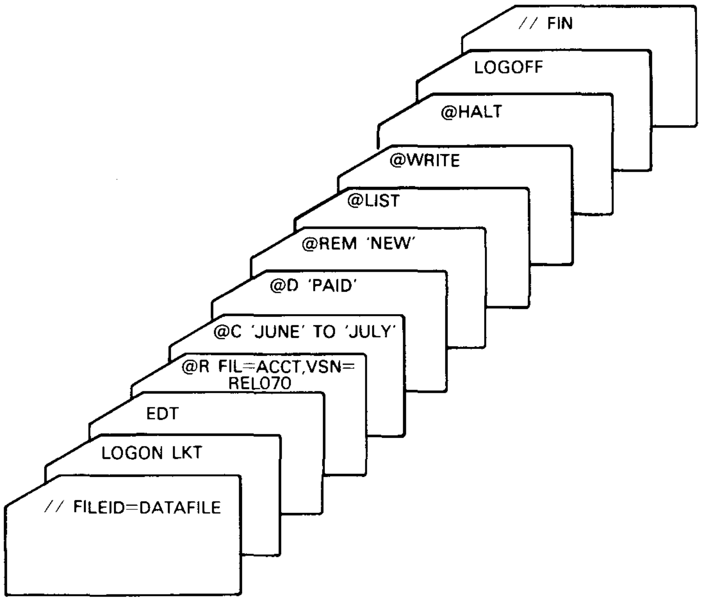


Figure 2-1. Batch EDT Session

Next, you store this routine (the card input) as a file on either the spool file via the INPUT reader command (which is described in the spooling and job accounting concepts and facilities manual) or on a library via the librarian's ELE command (which is discussed in the system service programs user guide).

After the routine is stored, you can run it by issuing the appropriate ENTER command from the system console. (The ENTER command is described in the interactive services commands and facilities user guide/programmer reference.)

3. EDT Commands

3.1. INTRODUCTION

EDT provides you with a special set of commands to create and update your files. With these commands you can manipulate anything in your files. You can add new information to your files, delete wrong or obsolete information from your files, or modify the existing contents of your files, from a single character up to the entire file.

You enter these commands from a workstation. You can enter them individually, or combine them with other commands to permit faster and more efficient editing. However, no matter which way you choose, you must always start each command line with the current command trigger (which is set to @ (an at sign) at the start of each new EDT session).

The following subsections explain the EDT commands. The commands are presented in an alphabetic sequence for easy reference. Each subsection contains the description, format, and some typical examples of a particular EDT command. For your convenience, the EDT commands are summarized in Table 3–1.

In line mode, you can use all the EDT commands. In screen mode, you can use all the EDT commands except the UPDATE command. No matter which mode you're in when you issue an EDT command, EDT returns you to that mode after it processes the command.

Table 3-1. EDT Commands

Command	Function
@	Sets the current line number and increment for data lines keyed in at the workstation.
CHANGE	Replaces an existing string in the current work-space file.
COPY	Copies lines in the current work-space file to new line locations without deleting the original lines.
DELETE	Erases specified lines from the current work-space file.
FIND	Locates the first occurrence of a string in the work-space file and assigns its corresponding line number to the variable ? and the column numbers of the first and last columns it occupies to [and] respectively.
FSTATUS	Creates in the current work-space file a list of all modules contained in a specified program library.
INSERT	Inserts a specified string into lines in the current work-space file.
LIST	Lists specified lines from the current work-space file on the printer.
MOVE	Transfers specified lines to new line locations in the current work-space file and deletes the original lines and line numbers.
NUMBER	Inserts sequence numbers into input lines.
PRINT	Displays specified lines from the current work-space file on the workstation screen.
PUNCH	Reproduces specified lines from the current work-space file on cards.
READ	Reads a copy of either a program library or data file residing on disk, diskette, tape, or cards into the work-space file.
REMOVE	Deletes a specified string from lines in the current work-space file.
SEQUENCE	Places sequence numbers in existing lines in the current work-space file.
UPDATE	Displays specified lines from the current work-space file one at a time for you to edit or change.
WRITE	Writes a copy of the current work-space file to a program library or data file on disk, diskette, or tape, or to the spool file.

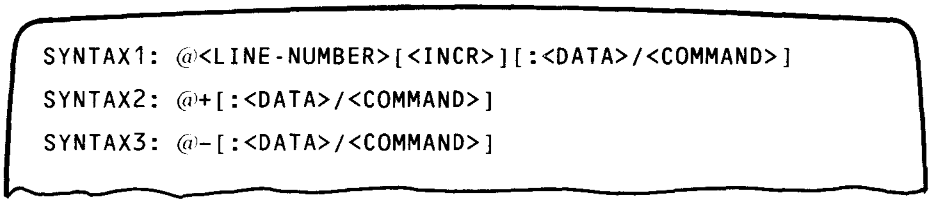
3.2. SETTING THE LINE NUMBER (@)

The @ command sets the current line number and increment for the data lines keyed in at the workstation.

The format is:

```
@⎧line-number[(incr)]⎫ ⎡:⎧data   ⎫⎤
 ⎨+                  ⎬ ⎣ ⎩command⎭⎦
 ⎩-                  ⎭
```

The help screen format is:



```
SYNTAX1: @<LINE-NUMBER>[<INCR>][:<DATA>/<COMMAND>]
SYNTAX2: @+[:<DATA>/<COMMAND>]
SYNTAX3: @-[:<DATA>/<COMMAND>]
```

where:

line-number
: Implicitly sets the current work-space line number to any number ranging from .0001 to 9999.9999. The current increment is implicitly set to 1, 0.1, 0.01, 0.001, or 0.0001, depending on the decimal location of the least significant digit specified in the line number.

incr
: Explicitly sets the current increment for subsequent line numbers. It is written in the form of nnnn.nnnn decimal digits. Leading and trailing zeros are optional but the parentheses are required. No space may exist between the line number and increment. If the incr parameter is omitted, the line number implicitly sets the increment.

+
: Causes EDT to access the current line plus the current increment in the work-space file.

-
: Causes EDT to access the current line minus the current increment in the work-space file.

`data`
Indicates data (text) that is being keyed in at the workstation.

`command`
Indicates commands that are being keyed in at the workstation.

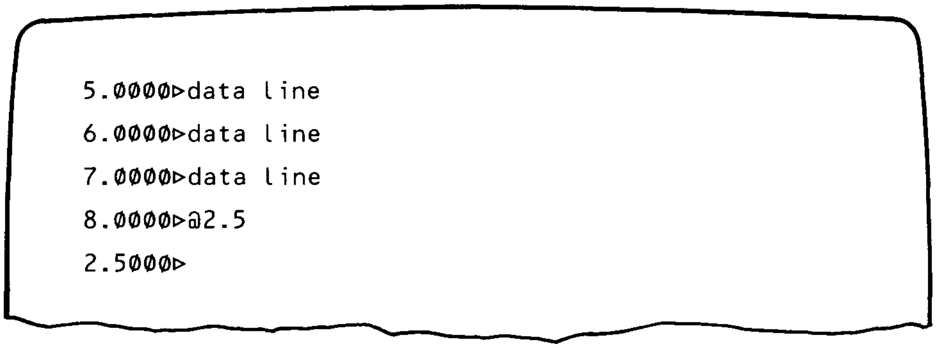
NOTES:

1. *If the current line number is incremented to 9999.0000 or greater by EDT, the current increment is automatically reset to 0.0001. This prevents the user from creating lines with line numbers greater than 9999.9999.*

2. *The increment and line number pertain only to the current work-space file.*

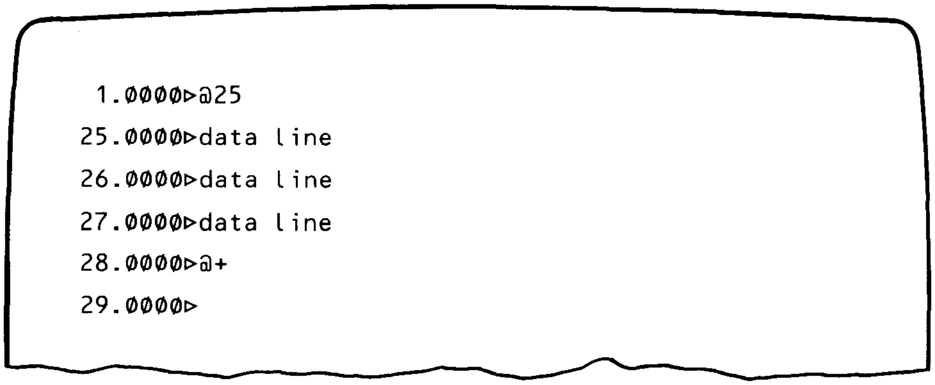
3. *If the line number is set to an already existing line, that line becomes the current line. If you enter anything on that line, EDT overwrites the old line.*

Some typical examples of the @ command are:



```
5.ØØØØ▷data line
6.ØØØØ▷data line
7.ØØØØ▷data line
8.ØØØØ▷@2.5
2.5ØØØ▷
```

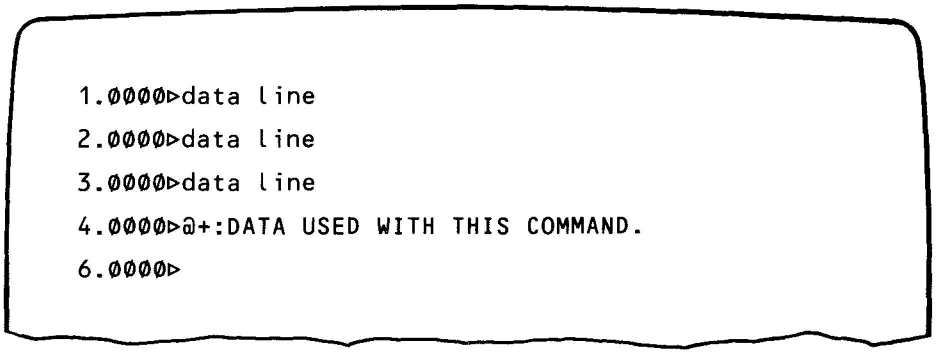
Here, the @2.5 command on line 8 sets the line number to 2.5000 and displays the line number on the next line. (Note that EDT doesn't display the contents of that line.)



```
 1.ØØØØ▷@25
25.ØØØØ▷data line
26.ØØØØ▷data line
27.ØØØØ▷data line
28.ØØØØ▷@+
29.ØØØØ▷
```

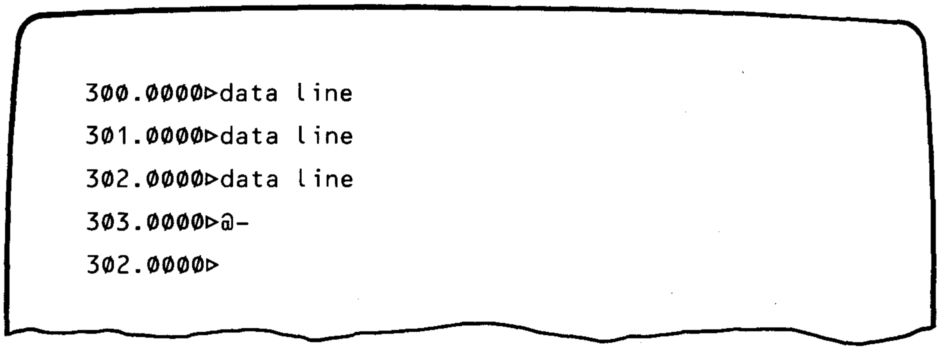
Here, the @+ command on line 28 displays the next line number (29) on the following line, which is ready to accept data or a command.

You can also key in data or commands as part of the @+ command as shown in the next example.



```
1.ØØØØ▷data line
2.ØØØØ▷data line
3.ØØØØ▷data line
4.ØØØØ▷@+:DATA USED WITH THIS COMMAND.
6.ØØØØ▷
```

Here, the @+ command on line 4 assigns the data keyed in with it to the next line number (5.0000). Notice that this line (5.0000) is not displayed, but the next line number following it (6.0000) is displayed. You can then key in data or a command.



```
3ØØ.ØØØØ▷data line
3Ø1.ØØØØ▷data line
3Ø2.ØØØØ▷data line
3Ø3.ØØØØ▷@-
3Ø2.ØØØØ▷
```

Here, the @– command on line 303 decreased the current line number by the current increment. The previous line number is displayed for you to key in data or a command. You can also key in data and commands with the @– command. The data is placed on the previous line and then the current line number is redisplayed for you to key in data or a command. Note that any data located on the previous line at the time the @– command is issued is overwritten.

The following table shows how the increment for work-space line numbers is automatically or explicitly set via the @ command.

Command	Line number set to	Increment set to
@15	15	1
@7.2	7.2	0.1
@312.05	312.05	0.01
@5.014	5.014	0.001
@10.0060	10.0060	0.0001
@4(.05)*	4.00	0.05

*Sets the increment explicitly to 0.05.

C

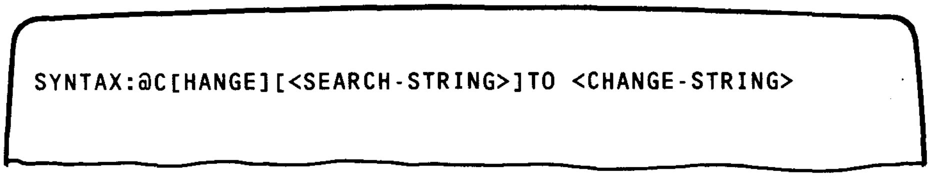
3.3. CHANGING DATA (@CHANGE)

The CHANGE command permits the changing of data on one or more lines in the work-space file. It replaces an existing string (search string) with a new string (change string). If the new string is not the same length as the previous one, the line is either increased or decreased.

The format is:

```
@CHANGE [ 'search-string'[ *n]] TO 'change-string'[ *n]
```

The help screen format is:



```
SYNTAX:@C[HANGE][<SEARCH-STRING>]TO <CHANGE-STRING>
```

where:

'search-string'
: Specifies a string in the current work-space file to be replaced. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. If no search string is specified, the change is performed as though the search string consisted of EXCLUDE characters the same length as the change string. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

'change-string'
: Specifies a string to be inserted in place of the search string in the current work-space file. The change string must be enclosed in apostrophes. Multiple copies of the change string may be specified by using the string multiplication factor *n, where n is the number of times the change string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

Some typical CHANGE command examples are:

```
@C 'BOB' TO 'ROBERT'
```

Changes the first occurrence of BOB on every line in the current work-space file to ROBERT. If BOB appears more than once on a line, the remaining occurrences are not changed.

```
@C TO 'ROBERT'
```

Changes the first six columns of every line in the current work-space file to ROBERT

```
@C '.'*10 TO '.'*5
```

Changes the first occurrence of .... (a separator between data fields, for example) to ....., thereby making room on every line in the current work-space file for another 5-character data field.

The following command modifiers can be added to the CHANGE command, increasing its capability: ALL, COLUMN, FIRST, and ON.

Commands that can be combined with the CHANGE command that permit faster and more efficient editing are: COPY, DELETE, FIND, FSTATUS, LIST, MOVE, PRINT, PUNCH, READ, UPDATE, and WRITE.

Some typical CHANGE examples using modifiers and other commands are:

```
@C ALL 'BOB' TO 'ROBERT'
```

Changes all occurrences of BOB to ROBERT in the current work-space file

```
@C 'BOB' TO 'ROBERT' LIST
```

Changes the first occurrence of BOB to ROBERT on every line in the current work-space file and lists on the printer the lines where the changes were made

```
@ON 10:20 COL 5:7 C TO 'BOB'
```

Changes columns 5 through 7 on lines 10 through 20 of the current work-space file to BOB

CO

3.4. COPYING LINES (@COPY)

The COPY command copies specified lines to new line locations in the work-space file without deleting the original lines. You can copy a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines.

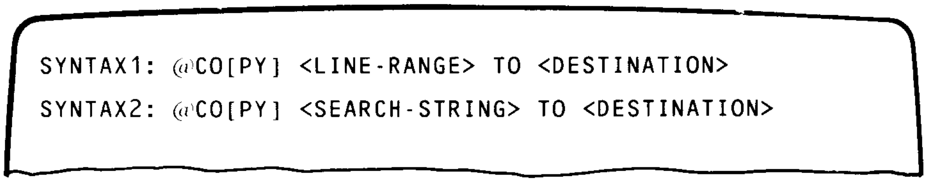
The format is:

```
@COPY [ line-range][ 'search-string'[ *n]] TO destination
```

NOTE:

At least one of the optional parameters must be specified.

The help screen format is:



```
SYNTAX1: @CO[PY] <LINE-RANGE> TO <DESTINATION>
SYNTAX2: @CO[PY] <SEARCH-STRING> TO <DESTINATION>
```

where:

line-range
: Specifies by line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be copied. Consecutive line ranges must be specified in ascending order.

'search-string'
: Specifies by a string in the lines those lines to be copied. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of apostrophes and quotes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

destination

Specifies by line number or line numbers the new line location or locations for copied lines. A single destination or multiple destinations may be specified.

When a single destination is specified, it's either the starting and ending point for a single line being copied (i.e., a line specified by a single line number or special symbol) or it's the starting point for multiple lines being copied (i.e., lines specified by a search string, line numbers, special symbols, or a combination of them).

When multiple destinations are specified: (1) a single line is copied to each specified destination, for example, @COPY 2 TO 32, 40 copies line 2 to lines 32 and 40, and (2) multiple lines are copied to successive lines beginning at each specified destination. For example, @COPY 5, 10, 15 TO 25, 50 copies lines 5, 10, and 15 to lines 25, 26, and 27 and to lines 50, 51, and 52 respectively.

Some typical COPY command examples are:

@CO 1 TO 5

Copies line 1 to line 5

@CO 2,6,9 TO 20

Copies lines 2, 6, and 9 to lines 20, 21, and 22 respectively. (Note that the implied increment in the destination is 1.)

@CO 1:9 TO 30(.2)

Copies lines 1 through 9 to lines 30, 30.2, 30.4, 30.6, etc. (Note that the increment of the destination is explicitly set to .2.)

@CO 1,3,5:9 TO 10(.3), 50

Copies lines 1, 3, and 5 through 9 first to lines 10, 10.3, 10.6, 10.9, etc and then to lines 50, 51, 52, 53, etc

@CO '.'*7 TO 10

Copies the first line containing .... (a separator between data fields, for example) to line 10 and any subsequent lines containing that string to lines 11, 12, 13, etc

@CO 'ACCOUNT 48' TO 15

Copies the first line containing ACCOUNT 48 to line 15 and any subsequent lines containing ACCOUNT 48 to lines 16, 17, 18, etc. (See note 3.)

@CO % TO 175

Copies the lowest line (%) in the current work-space file to line 175

@CO 57 'PAYROLL' TO 200

Copies line 57 to line 200 if it contains PAYROLL

NOTES:

1. *Lines are copied in the order that they are specified in the command. In the case of a search string (for instance, all lines containing ACCOUNT 48), lines are copied in the order that they appear in the work-space file.*

2. *Remember that if lines are copied to destinations beyond the highest line in the current work-space file, the current work-space line number is reset to the highest destination line plus the current increment.*

3. *When you copy all lines specified by a search string only, make sure that you copy them to a line destination lower than the first occurrence of the search string; otherwise, a program loop will develop. If, however, you specify a higher destination and a loop does develop, press F2 (function key 2) to terminate it.*

The following command modifiers can be added to the COPY command, increasing its capability: COLUMN, FIRST, and NOT.

Commands that can be combined with the COPY command that permit faster and more efficient editing are: CHANGE, DELETE, FIND, FSTATUS, INSERT, LIST, PRINT, PUNCH, READ, REMOVE, SEQUENCE, UPDATE, and WRITE.

Some typical COPY examples using modifiers and other commands are:

```
@CO FIRST 'PENNY' TO 85
```

Copies the first line containing PENNY in the current work-space file to line 85

```
@CO 7:10 TO 90 PRINT
```

Copies lines 7 through 10 to lines 90, 91, 92, etc and displays the newly copied lines

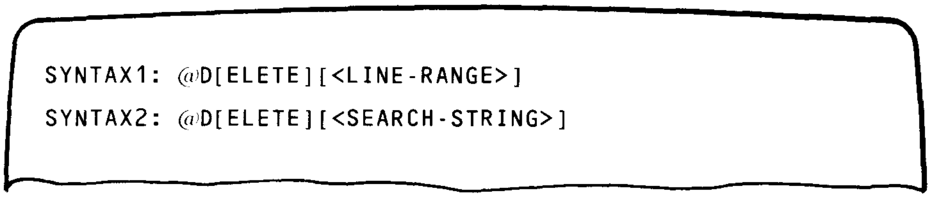
3.5. DELETING LINES (@DELETE)

The DELETE command erases specified lines from the current work-space file. You can erase a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines.

The format is:

```
@DELETE [line-range][ 'search-string'[ *n]]
```

The help screen format is:



```
SYNTAX1: @D[ELETE][<LINE-RANGE>]
SYNTAX2: @D[ELETE][<SEARCH-STRING>]
```

where:

`line-range`

Specifies by line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, or any combination of these lines to be deleted. Consecutive line ranges must be specified in ascending order.

`'search-string'`

Specifies by a string in the lines those lines to be deleted. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

NOTE:

If you use only the @D command all lines in the current work-space file are deleted.

Some typical examples of the DELETE command are:

```
@D 5
```
Erases line 5

```
@D 1,4,7,9
```
Erases lines 1, 4, 7, and 9

```
@D 10:15
```
Erases lines 10 through 15

```
@D 'DIRECTORY'
```
Erases every line containing DIRECTORY

```
@D '.'*10
```
Erases every line containing .... as a separator between data, for example.

```
@D %
```
Erases the lowest line (%) in the current work-space file

```
@D 3,8 'ITEM 10'
```
Erases line 3 if it contains ITEM 10 and erases line 8 if it contains ITEM 10

The following command modifiers can be added to the DELETE command, altering its capability: COLUMN, FIRST, NOT, and ON.

Commands that can be combined with the DELETE command that permit faster and more efficient editing are: CHANGE, COPY, FIND, INSERT, PRINT, REMOVE, SEQUENCE, and UPDATE.

Some typical DELETE examples using modifiers and other commands are:

```
@LIST ALL 'TRUCKS' D
```
Prints all lines containing TRUCKS and then deletes them

```
@FIND 'CARS' D
```
Locates the first line containing CARS in the current work-space file and deletes that line

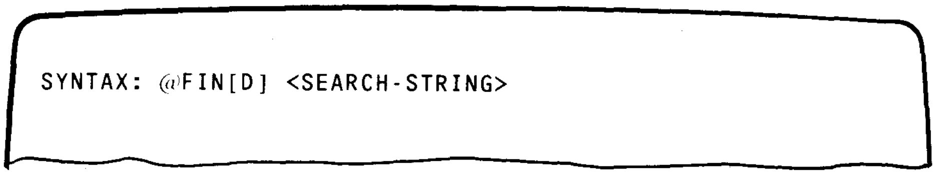
3.6. LOCATING DATA (@FIND)

The FIND command locates the first occurrence of a specified string in the work-space file and assigns its corresponding line number to the variable symbol ?, and it assigns the column numbers of the first and last columns the string occupies to the left bracket ([) and the right bracket (]) respectively. The line containing the specified string is not displayed unless the FIND command is combined with the PRINT command.

The format is:

```
@FIND 'search-string'[*n]
```

The help screen format is:



```
SYNTAX: @FIN[D] <SEARCH-STRING>
```

where:

search-string
: Specifies a string to be located in the work-space file. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes, or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

The purpose of assigning line and column numbers to variables is that it enables you to execute other EDT commands against a line containing a particular string (or the string itself) without knowing the corresponding line or column numbers. Instead, you know the variable that stands for the line or column number.

Some typical examples of the FIND command are:

@FIN 'CURRENT BALANCES'

Locates the first occurrence of CURRENT BALANCES in the work-space file. If, for example, it appears on line 8 in columns 5 through 20, ? is set to 8, [is set to 5, and] is set to 20.

@FIN '.'*10

Locates the first occurrence of .... (a separator between data fields, for example) in the work-space file. If, for example, it appears on line 10 in columns 12 through 21, ? is set to 10, [is set to 12, and] is set to 21.

NOTE:

Once the variables ?, [, and] are defined by the FIND command, they retain those values throughout the EDT session until another successful search operation (a command containing another search string) is issued to change them.

The following command modifiers can be added to the FIND command, altering its capability: ALL, COLUMN, FIRST, NOT, and ON.

Commands that can be combined with the FIND command that permit faster and more efficient editing are: CHANGE, COPY, DELETE, FSTATUS, INSERT, LIST, MOVE, NUMBER, PRINT, PUNCH, READ, REMOVE, SEQUENCE, UPDATE, and WRITE.

Some typical FIND examples using modifiers and other commands are:

```
@FIN 'JOE SMITH' MOVE TO 25
```

Locates the first occurrence of JOE SMITH and moves the line containing it to line 25

```
@ON 8 FIN 'SUSAN MORRIS'
```

Locates the first occurrence of SUSAN MORRIS on line 8. If, for example, the string appears in columns 10 through 21, ? is assigned the value 8 and [and] are assigned the values 10 and 21 respectively.

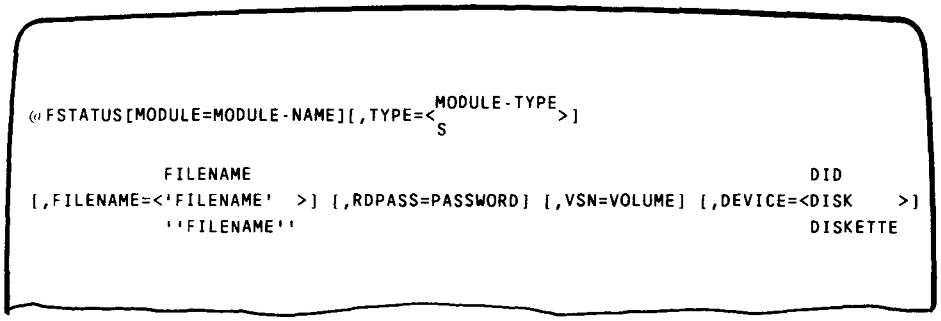
3.7. LISTING MODULE NAMES AND TYPES (@FSTATUS)

The FSTATUS command creates in the work-space file a list of all modules contained in a specified program library. The list of modules is created using the current line and increment in the work-space file. Each line of the list contains the module name, module type, creation date, creation time, and comment field.

Format:

```
@FSTATUS [MODULE=module-name] [,TYPE={module-type}]
                              [      {S          }]

      ,FILENAME={filename    }[,RDPASS=password],VSN=volume[,DEVICE={did     }]
                {'filename'  }                             [        {DISK    }]
                {''filename''}                             [        {DISKETTE}]
```

The help screen format is:



```
@FSTATUS [MODULE=MODULE-NAME] [,TYPE=<MODULE-TYPE>]
                                     S

           FILENAME                                                        DID
[,FILENAME=<'FILENAME'   >] [,RDPASS=PASSWORD] [,VSN=VOLUME] [,DEVICE=<DISK      >]
           ''FILENAME''                                                    DISKETTE
```

NOTE:

You will not immediately see this help screen when you first issue the @PROMPT command to get help with the @FSTATUS command. To see this help screen, you must press FK13 for additional help.

where:

```
MODULE=module-name
```

Specifies the name of a specific module to be listed. It can be from one to eight alphanumeric characters with the first character being alphabetic. If you specify a module name, EDT lists information about that module only.

```
TYPE={module-name}
     {S          }
```

Specifies the type of module to be read. For SAT files, the types permitted are source (S), macro (M), procedure (P), load (L), object (O), or proc-name (PN). For MIRAM files, the types permitted are screen format (F) or (FC), help screen (HELP), saved run library (J), or menu (MENU).

```
FILENAME={filename    }
         {'filename'  }
         {''filename''}
```

Specifies the name of the program library. It can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, parentheses, or commas are used in it.

```
RDPASS=password
```

Specifies a read password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

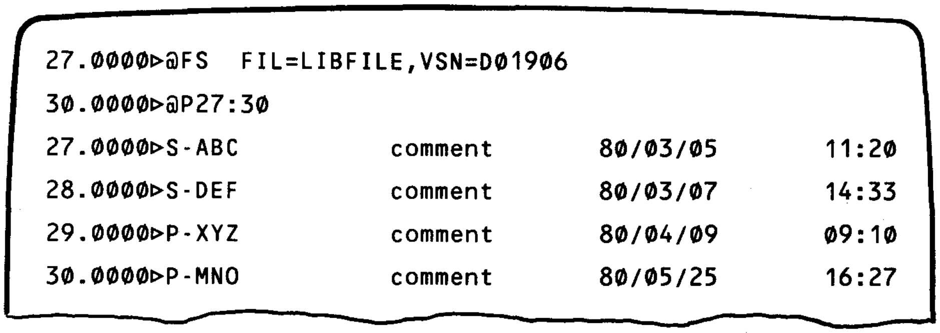
```
VSN=volume
```

Specifies the volume serial number of the volume containing the file. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously entered the file in the system file catalog. If you don't know the vsn of the volume containing the file, you can specify the DEVICE parameter instead.

```
DEVICE={did     }
       {DISK    }
       {DISKETTE}
```

Specifies the device where the file resides. did specifies the device address (did) of the device. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISK and DISKETTE, specify the type of device, but not the specific device. If you do not specify a device, the parameter defaults to the disk or diskette having the vsn specified in the VSN parameter.

A typical example of the FSTATUS command is:



```
27.ØØØØ▷@FS  FIL=LIBFILE,VSN=DØ19Ø6
3Ø.ØØØØ▷@P27:3Ø
27.ØØØØ▷S-ABC           comment       8Ø/Ø3/Ø5        11:2Ø
28.ØØØØ▷S-DEF           comment       8Ø/Ø3/Ø7        14:33
29.ØØØØ▷P-XYZ           comment       8Ø/Ø4/Ø9        Ø9:1Ø
3Ø.ØØØØ▷P-MNO           comment       8Ø/Ø5/25        16:27
```

Here, the FSTATUS command on line 27 produces a list of the modules contained in the library LIBFILE. Notice that LIBFILE contains the source modules ABC and DEF and the procedure modules XYZ and MNO.

Commands that can be combined with the FSTATUS command that permit faster and more efficient editing are: CHANGE, COPY, FIND, INSERT, REMOVE, and SEQUENCE.

A typical example of FSTATUS combined with other commands is:

@FS FIL='FILEJ',VSN=D00028 SEQ * COL 100

Inserts sequence numbers starting with the current work-space line number (*) in columns 100 through 108 for each directory (module name and type) listed for the file FILE J

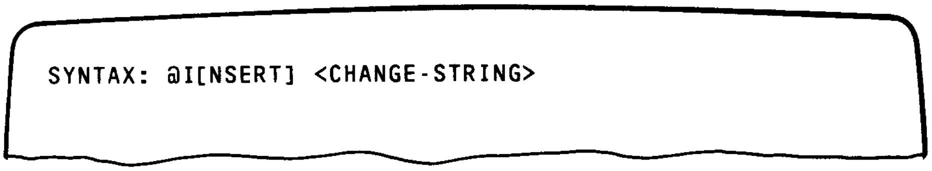
3.8. INSERTING DATA (@INSERT)

The INSERT command inserts a specified string into lines in the current work-space file. Existing characters are shifted to the right to insert the string. If the insertion causes the line to exceed the maximum line length, the line is truncated to the maximum length.

The format is:

```
@INSERT 'change-string'[ *n]
```

The help screen format is:



```
SYNTAX: @I[NSERT] <CHANGE-STRING>
```

where:

`'change-string'`
Specifies a string to be inserted into the work-space file. This string must be enclosed in apostrophes. Multiple copies of the change string may be specified by using the string multiplication factor *n, where n is the number of times the original change string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

Unless otherwise specified, the string is inserted beginning in column 1 on every line. Existing characters are shifted to the right (the length of inserted string) to incorporate the insert.

Some typical examples of the INSERT command are:

@I 'ABC'
Inserts A in column 1, B in column 2, and C in column 3 on every line in the current work-space file

@I '.'*10
Inserts .... (a separator between data fields, for example) in columns 1 through 10 on every line in the current work-space file.

The following command modifiers can be added to the INSERT command, altering its capability: COLUMN, FIRST, and ON.

Commands that can be combined with the INSERT command that permit faster and more efficient editing are: COPY, DELETE, FSTATUS, LIST, MOVE, PRINT, PUNCH, READ, UPDATE, and WRITE.

Some typical INSERT examples using modifiers and other commands are:

```
@COL 7 I 'YEAR'
```

Inserts YEAR in columns 7 through 10 on every line in the current work-space file

```
@ON 4 COL 12:15 I '1980' P
```

Inserts 1980 in columns 12 through 15 on line 4, and then displays line 4

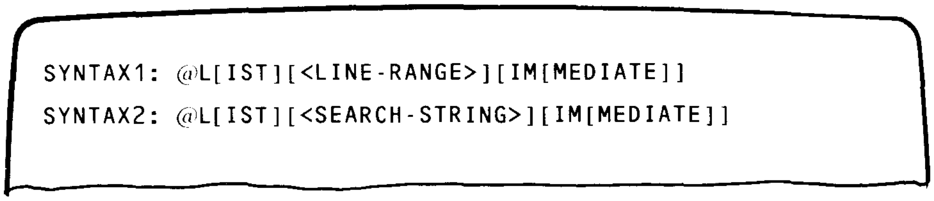
3.9. LISTING LINES ON THE PRINTER (@LIST)

The LIST command lists specified lines from the current work-space file on the printer. You can list a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines. The @LIST command does not list the work-space line numbers or any EDT commands you may have used while entering the lines in the current work-space file.

The format is:

```
@LIST [line-range]['search-string'[*n]] [IMMEDIATE]
```

The help screen format is:



```
SYNTAX1: @L[IST][<LINE-RANGE>][IM[MEDIATE]]
SYNTAX2: @L[IST][<SEARCH-STRING>][IM[MEDIATE]]
```

where:

line-range
: Specifies by line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be printed. Consecutive line ranges must be specified in ascending order.

search-string
: Specifies by a string in the lines those lines to be printed. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

IMMEDIATE
: Indicates that the data lines specified in the LIST command are listed immediately on the printer. If IMMEDIATE is not specified, the lines aren't listed until the end of the EDT session.

NOTE:

Using only the @L command without any parameters causes all lines in the current work-space file to be listed on the printer.

Some typical examples of the list command are:

```
@L 9
```
Lists line 9 on the printer

```
@L 2,4,7
```
Lists lines 2, 4, and 7 on the printer

```
@L 20:25
```
Lists lines 20 through 25 on the printer

```
@L 'REPORT'
```
Lists all lines containing REPORT on the printer

```
@L 3,11:13, $
```
Lists line 3, lines 11 through 13, and the highest line in the current work-space file ($) on the printer

```
@L '.'*10 IM
```
Lists on the printer all lines containing .... (a separator between data fields, for example). Lines are listed immediately, rather than at the end of the EDT session.

NOTE:

Lines are listed in the order that they are specified in the command; when specifying a search string, lines are listed in the order that they appear in the work-space file.

The following command modifiers can be added to the LIST command, altering its capability: NOT, ON, and SHOW.

Commands that can be combined with the LIST command that permit faster and more efficient editing are: CHANGE, COPY, FIND, INSERT, REMOVE, and SEQUENCE.

Some typical examples of the LIST command using modifiers and other commands are:

```
@FIND 'QUESTION' L
```
Locates the first occurrence of QUESTION in the work-space file and lists the line containing it on the printer

```
@ON 1:10 L 'FALSE'
```
Lists on the printer any line from 1 to 10 that contains FALSE

M

3.10. MOVING LINES (@MOVE)

The MOVE command transfers specified lines to new line locations (destinations) in the work-space file and erases the original lines and line numbers. You can move a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines. The MOVE command is actually a COPY command followed by a DELETE command. Each line is deleted immediately after being copied.

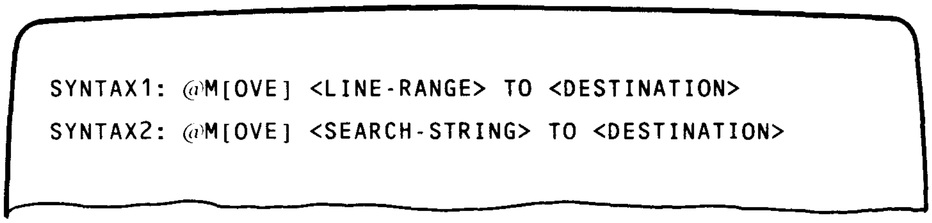
The format is:

```
@MOVE [line-range]['search-string'[*n]] TO destination
```

NOTE:

At least one of the optional parameters must be specified.

The help screen format is:



```
SYNTAX1: @M[OVE] <LINE-RANGE> TO <DESTINATION>
SYNTAX2: @M[OVE] <SEARCH-STRING> TO <DESTINATION>
```

where:

`line-range`
: Specifies by line number or line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be moved. Consecutive line ranges must be specified in ascending order.

`'search-string'`
: Specifies by a string in the lines those lines to be moved. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

destination

Specifies by a line number the starting point for the new line location or locations for moved lines. Only a single destination may be specified.

When moving a single line, the destination line number is the starting and ending point of the moved line. However, the destination line number is the starting point for multiple lines being moved (i.e., lines specified by a search string, line range, special symbols, or a combination of them).

Some typical examples of the MOVE command are:

```
@M 6 TO 12
```

Moves line 6 to line 12

```
@M 4,8,12 TO 20
```

Moves lines 4, 8, and 12 to lines 20, 21, and 22 respectively

```
@M 10:20 TO 30(.4)
```

Moves lines 10 through 20 to lines 30, 30.4, 30.8, etc. Notice that the increment for the line destinations is explicitly set to .4

```
@M 'PRICES' TO 10
```

Moves all the lines containing PRICES in the current work-space file to lines 10, 11, 12, etc. (See note 3.)

```
@M % 'COST' TO 500
```

Moves the lowest line (special symbol %) in the current work-space file to line 500 if it contains COST

```
@M '.'*10 TO 15
```

Moves all lines containing .... (a separator between data fields, for example) to lines 15, 16, 17, etc

NOTES:

1. *Lines are moved in the order that they are specified in the command; when specifying a search string, lines are moved in the order that they appear in the work-space file.*

2. *Remember that if lines are moved to destinations beyond the highest line in the current work-space file, the current line number is set to the highest destination plus the current increment.*

3. *When you move all lines specified by a search string only, make sure that you move them to a line destination lower than the first occurrence of the search string; otherwise, a program loop will develop. If, however, you specify a higher destination and a loop does develop, press F2 (function key 2) to terminate it.*

The following command modifiers can be added to the MOVE command, increasing its capability: COLUMN, FIRST, NOT, and ON.

Commands that can be combined with the MOVE command that permit faster and more efficient editing are: CHANGE, FIND, INSERT, PRINT, REMOVE, SEQUENCE, and UPDATE.

Some typical MOVE examples using modifiers and other commands are:

```
@COLUMN 50 M 'BILLS' TO 7.01
```

Moves all lines containing BILLS starting in column 50 to lines 7.01, 7.02, 7.03, etc

```
@M 4 TO 24 UPDATE
```

Moves line 4 to line 24 and then displays line 24 to be updated via the UPDATE command

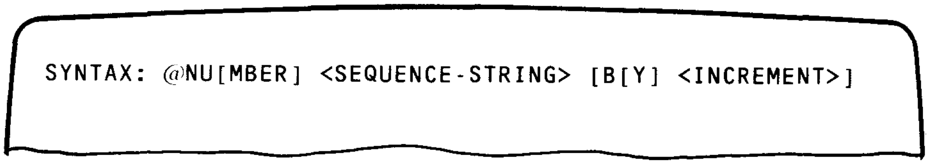
3.11. INSERTING SEQUENCE NUMBERS INTO INPUT LINES (@NUMBER)

The NUMBER command inserts sequence numbers into subsequent data lines keyed in at the workstation. The number command lets you sequence data.

The format is:

```
@ NUMBER 'sequence-string'[*n] [BY increment]
```

The help screen format is:



```
SYNTAX: @NU[MBER] <SEQUENCE-STRING> [B[Y] <INCREMENT>]
```

where:

'sequence-string'
: Specifies an alphanumeric string that is to be inserted on the next line keyed in at the workstation. The sequence string must be enclosed in apostrophes and one or more of its rightmost characters must be an integer from 0 to 9. A sequence string of up to 15 characters may be specified. Multiple copies of the sequence string may be specified by using the string multiplication factor *n, where n is the number of times the original sequence string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

increment
: Sets the increment for sequence strings inserted into input lines. A number of up to nine decimal digits may be specified as the increment.

Unless tabs have been set for the EDT session, the NUMBER command in its basic format inserts sequence numbers beginning in column 1 on subsequent data lines. If tabs have been set, EDT processes the tabs first before inserting the sequence numbers.

Sequence numbers do not appear on the screen as you enter your data. However, if you want to see the sequence numbers included in the lines, issue the appropriate PRINT command to display them on the workstation screen.

Sequence numbering continues until you enter the next EDT command.

A typical example of the NUMBER command is:

```
@NU 'EXT200' BY 5
```

Inserts the sequence string 'EXT200' in columns 1 through 6 on the next line keyed in at the workstation and places increments of 5 of that sequence string on subsequent lines keyed in. EXT205 will be inserted on the second line keyed in, EXT210 on the third line, EXT215 on the fourth line, and so forth.

The only command modifier that can be added to the NUMBER command is COLUMN. It enables you to insert sequence numbers starting in a column other than 1.

Commands that can be combined with the NUMBER command that permit faster and more efficient editing are: PRINT and UPDATE.

Some typical NUMBER examples using modifiers and other commands are:

```
@NU '0001' BY 0001 COL 77
```

Inserts sequence numbers in columns 77 through 80 on input lines beginning with the sequence string 0001 and incrementing it by 0001

```
@NU 'X500' BY 5 PRINT
```

Inserts sequence numbers in columns 1 through 4 on input lines (starting with the sequence string X500 and incrementing it by 5) and displays the lines

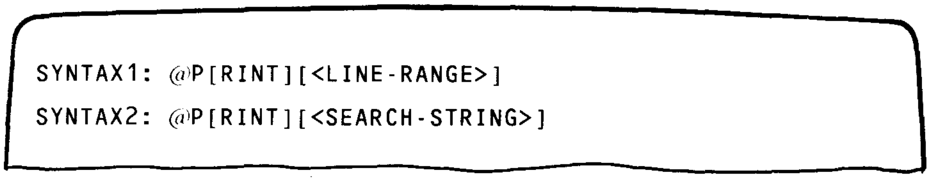
3.12. DISPLAYING LINES (@PRINT)

The PRINT command displays specified lines from the work-space file on the workstation screen. You can display a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines.

The format is:

```
@PRINT [ line-range][ 'search-string'[ *n] ]
```

The help screen format is:



```
SYNTAX1: @P[RINT][<LINE-RANGE>]
SYNTAX2: @P[RINT][<SEARCH-STRING>]
```

where:

`line-range`
: Specifies by line numbers, special symbols, and EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be displayed. Consecutive line ranges must be specified in ascending order.

`'search-string'`
: Specifies by a string in the lines those lines to be displayed. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the string can be specified through the use of the string multiplication factor *n, where n is the number of times the string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

EDT displays the lines you request with the PRINT command according to your specification for the SCRDSPLY parameter on the @SET directive. If SCRDSPLY=TRUNCATE, EDT displays one record per one screen line and truncates lines if necessary; if SCRDSPLY=FOLD, EDT displays one record per two screen lines and doesn't truncate lines.

After EDT prints a full screen or the last screen, you have four options:

1. You can update the displayed lines and transmit them to the EDT work space.

2. You can display remaining specified lines by pressing the F19 function key.

3. You can prevent the display of succeeding lines by pressing function key F1 or F18.

4. You can terminate the entire PRINT command by pressing function key F2.

For more information on the use of these and other function keys, see Section 9 and the interactive services commands and facilities user guide/programmer reference.

NOTE:

If you use the PRINT command without any parameters, all the lines in the current work-space file are displayed.

Some typical examples of the PRINT command are:

```
@P 23
```

Displays line 23 on the workstation screen

```
@P 3,16
```

Displays lines 3 and 16 on the workstation screen

```
@P 4:9
```

Displays lines 4 through 9 on the workstation screen

```
@P 'TIME'
```

Displays on the workstation screen all lines in the current work-space file that contain TIME

```
@P 2, 24:28, 32 'MEMBERS'
```

Displays on the workstation screen line 2, lines 24 through 28, or line 32 if the lines contain MEMBERS

```
@P '.'*10
```

Displays on the workstation screen all lines in the current work-space file containing .... (a separator between data fields, for example)

Lines are displayed in the order specified in the command; when specifying a search string, lines are displayed in the order that they appear in the work-space file.

The following command modifiers can be added to the PRINT command, altering its capability: ALL, COLUMN, NOT, ON, and SHOW.

Commands that can be combined with the PRINT command that permit faster and more efficient editing are: CHANGE, COPY, DELETE, FIND, INSERT, MOVE, NUMBER, REMOVE, SEQUENCE, and UPDATE.

Some typical PRINT examples using modifiers and other commands are:

```
@CHANGE ALL 'APPLES' TO 'PLUMS' P
```

Changes every occurrence of APPLES to PLUMS in the current work-space file and then displays all the lines where the changes were made

```
@FIN 'PEACHES' P
```

Locates the first occurrence of PEACHES in the work-space file and displays that line

PU

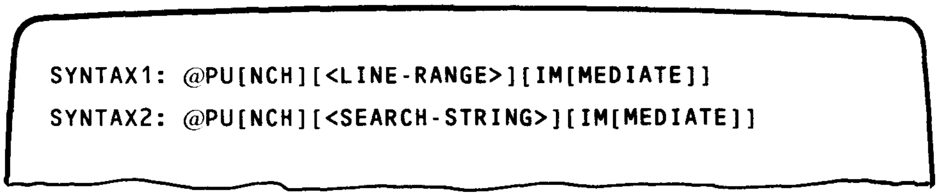
3.13. PUNCHING LINES (@PUNCH)

The PUNCH command reproduces specified lines from the work-space file on cards. You can punch a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines.

The format is:

```
@PUNCH [line-range][ 'search-string'[*n]][IMMEDIATE]
```

The help screen format is:



```
SYNTAX1: @PU[NCH][<LINE-RANGE>][IM[MEDIATE]]
SYNTAX2: @PU[NCH][<SEARCH-STRING>][IM[MEDIATE]]
```

where:

`line-range`
: Specifies by line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be punched. Consecutive line ranges must be specified in ascending order.

`'search-string'`
: Specifies by a string in the lines those lines to be punched. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the multiplication factor can't exceed 50.

`IMMEDIATE`
: Indicates that the data lines specified in the PUNCH command are to be punched immediately. If IMMEDIATE is not specified, the lines aren't punched until the end of the EDT session.

NOTE:

Using only the PUNCH command without any parameters (@PU) causes all lines in the current work-space file to be punched.

Some typical examples of the PUNCH command are:

```
@PU 5
```

Punches line 5 on a card

```
@PU 1,6,8
```

Punches lines 1, 6, and 8 on cards

```
@PU 2:5
```

Punches lines 2 through 5 on cards

```
@PU 'CHANNEL'
```

Punches on cards all lines in the current work-space file containing CHANNEL

```
@PU '.'*10 IM
```

Punches on cards all lines in the current work-space file containing ....(a separator between data fields, for example). Lines are punched immediately, rather than at the end of the EDT session.

```
@PU 4, 9:11, 13:15, $
```

Punches line 4, lines 9 through 11, lines 13 through 15, and the highest line ($) in the current work-space file on cards

```
@PU 'SPORTS' 7,9
```

Punches line 7 on a card if it contains SPORTS and also punches line 9 on a card if it contains SPORTS

Lines are punched in the order specified in the command; when specifying a search string, lines are punched in the order that they appear in the work-space file.

The following command modifiers can be added to the PUNCH command, altering its capability: COLUMN, NOT, and SHOW.

Commands that can be combined with the PUNCH command that permit faster and more efficient editing are: CHANGE, COPY, FIND, INSERT, REMOVE, and SEQUENCE.

Some PUNCH examples using modifiers and other commands are:

```
@CHANGE 'CARS' TO 'TRUCKS' PU
```

Changes the first occurrence of CARS on every line in the current work-space file to TRUCKS and then punches the changed lines on cards

```
@PU SHOW 20:30
```

Punches only columns 20 through 30 of each line in the work-space file on cards

R

3.14. READING DATA (@READ)

The READ command reads a copy of either a program library module or a data file residing on disk, diskette, tape, or cards into the work-space file.

The READ command is divided into seven basic formats. The formats are used to read to the current work-space file:

a SAT* or MIRAM** library module on a disk or format label diskette;

a MIRAM data file on a disk or format label diskette;

a unit record file from a data set label diskette or card reader;

a file from a tape;

a file from the spool file;

the same module or file last accessed through a previous @READ or @WRITE command; or

the same module or file last accessed through a previous @READ or @WRITE command but read now with a KEY, KKEY, or SHOW parameter or any valid EDT command specified.

The following paragraphs describe these seven formats and give examples for each. Here is some general information about the READ command that pertains to all seven formats:

- You can enter the READ command via keywords or positional parameters. We discuss the keyword parameters here because they're used most often. For information on how you can enter the READ command via positional parameters, see Appendix E.

- Unlike the keywords in the other EDT commands, the keywords in the READ command may not be separated by spaces. If you leave spaces between them, EDT can't process the command and you'll receive an error message.

- Commands that you can combine with the READ command for more efficient and faster editing are: CHANGE, COPY, FIND, INSERT, MOVE, SEQUENCE, and UPDATE. We provide examples of the @READ command in combination with other commands after the descriptions of the five formats so that you're familiar with the various parameters associated with the READ command when studying the examples.

** SAT means system access technique*

***MIRAM means multiple indexed random access method*

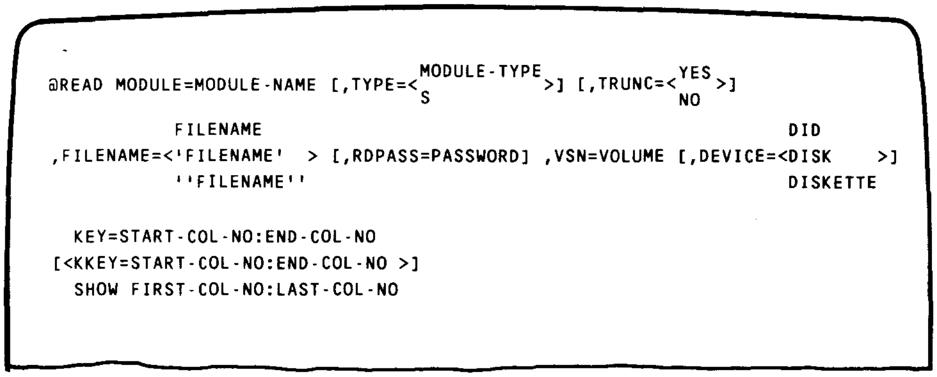
- As we explained in the statement of conventions, we show the help screen formats you'll see when you issue an @PROMPT screen command along with the formats we present for the @READ command. You should note that you will not immediately see the help screens for the seven formats of the @READ command when you first issue the @PROMPT command to get help with the @READ command. To see help screens for the seven formats, you must follow the procedure for getting additional help from an @PROMPT command help screen. That is, you must press F13 for additional help until you see the format for the type of file you want to read. We describe the procedure for getting additional help from an @PROMPT command help screen in full detail in Section 8.

Format 1

To read a SAT or MIRAM library module from a disk or format label diskette to the current work-space file, use this format:

```
@READ MODULE=module-name[,TYPE={module-type}][,TRUNC={YES}]
                               {S          }          {NO }
    ,FILENAME={filename    }[,RDPASS=password],VSN=volume[,DEVICE={did     }]
              {'filename'  }                                      {DISK    }
              {''filename''}                                      {DISKETTE}
    Δ[{KEY=start-col-no:end-col-no  }]
     [{KKEY=start-col-no:end-col-no }]
     [{SHOWΔfirst-col-no:last-col-no}]
```

The help screen format is:



```
                                      MODULE-TYPE                YES
@READ MODULE=MODULE-NAME [,TYPE=<                >] [,TRUNC=<        >]
                                      S                          NO

          FILENAME                                                   DID
,FILENAME=<'FILENAME'  > [,RDPASS=PASSWORD] ,VSN=VOLUME [,DEVICE=<DISK     >]
          ''FILENAME''                                               DISKETTE

  KEY=START-COL-NO:END-COL-NO
[<KKEY=START-COL-NO:END-COL-NO >]
  SHOW FIRST-COL-NO:LAST-COL-NO
```

where:

MODULE=module-name

Specifies the name of the module to be read. It can be from one to eight alphanumeric characters with the first character being alphabetic.

```
TYPE={module-name}
     {S          }
```

Specifies the type of module to be read. For SAT files, the types permitted are source (S), macro (M), procedure (P), load (L), object (O), or proc-name (PN). For MIRAM files, you may specify screen format (F) or (FC); help screen (HELP); saved run library (J); or menu (MENU).

```
TRUNC={YES}
      {NO }
```

Specifies whether input record truncation should be reported. If you omit this keyword or specify NO, EDT truncates records but displays no warning message. This parameter applies only to MIRAM library modules, not SAT library modules.

```
FILENAME={filename    }
         {'filename'  }
         {''filename''}
```

Specifies the name of the program library containing the module. The file name is the LBL name of the file and can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, parentheses, or commas are used in it.

```
RDPASS=password
```

Specifies a read password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

```
VSN=volume
```

Specifies the volume serial number of the file where the module resides. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously entered the file in the system file catalog. If you do not know the vsn of the disk or diskette your module resides on, you can specify the device address parameter (DEVICE) instead.

```
DEVICE={did     }
       {DISK    }
       {DISKETTE}
```

Specifies the device you want to read from. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISK and DISKETTE, specify the type of device your module resides on, but not the specific device. If you do not specify a device, the parameter will default to the disk or diskette having the volume serial number you specified for the VSN parameter.

```
KEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by KEY must contain decimal characters.

```
KKEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by KKEY must contain decimal characters.

```
SHOW△first-col-no:last-col-no
```

Specifies the first and last column numbers of data to be used as input records. Data is read into the work-space file using the current line number and increment.

NOTE:

For more detailed information on the use of KEY, KKEY, and SHOW, refer to format 2 of the READ command.

A typical example of reading a library module is:

```
@R MO=MYPROG,FIL=LIBFILE,RD=PUBS,VSN=DØØØ31
```

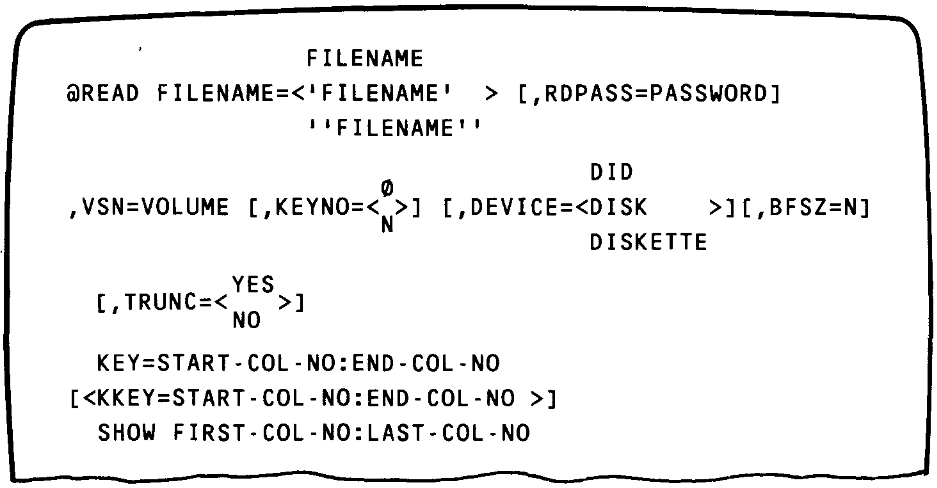
This command reads into the current work-space file a copy of the library module MYPROG from the file LIBFILE on disk volume D00031. We specify the read password PUBS because the file was previously entered in the system file catalog with it.

Format 2

To read a MIRAM data file from a disk or format label diskette to the current work-space file, use this format:

```
@READ FILENAME=⎧filename    ⎫[,RDPASS=password]
               ⎨'filename'  ⎬
               ⎩''filename''⎭

     ,VSN=volume⎡,KEYNO=⎧Ø⎫⎤⎡,DEVICE=⎧did     ⎫⎤[,BFSZ=n]
                ⎣       ⎩n⎭⎦⎢        ⎨DISK    ⎬⎥
                            ⎣        ⎩DISKETTE⎭⎦

     ⎡,TRUNC=⎧YES⎫⎤
     ⎣       ⎩NO ⎭⎦

     △⎡⎧KEY=start-col-no:end-col-no  ⎫⎤
      ⎢⎨KKEY=start-col-no:end-col-no ⎬⎥
      ⎣⎩SHOW△first-col-no:last-col-no⎭⎦
```

The help screen format is:



```
                     FILENAME
 @READ FILENAME=<'FILENAME'  > [,RDPASS=PASSWORD]
                     ''FILENAME''
                           Ø                DID
 ,VSN=VOLUME [,KEYNO=< >] [,DEVICE=<DISK     >][,BFSZ=N]
                           N                DISKETTE

            YES
   [,TRUNC=<   >]
            NO

   KEY=START-COL-NO:END-COL-NO
 [<KKEY=START-COL-NO:END-COL-NO >]
   SHOW FIRST-COL-NO:LAST-COL-NO
```

where:

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the name of the data file to be read. The file name is the LBL name of the file and can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, parentheses, or commas are used in it.

RDPASS=password

Specifies a read password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

VSN=volume

Specifies the volume serial number of the file to be read. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously entered the file in the system file catalog. If you do not know the vsn of the disk or diskette your file resides on, you can specify the device address parameter (DEVICE) instead.

```
KEYNO=⎧Ø⎫
      ⎩n⎭
```

Specifies the number of the key to be used to sequentially read a MIRAM file. Keys are read in an alphanumeric sequence (A through Z and then 0 through 9). n may equal 1 to 5.

DEVICE= $\begin{Bmatrix}\text{did}\\\text{DISK}\\\text{DISKETTE}\end{Bmatrix}$

Specifies the device you want to read from. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISK and DISKETTE, specify the type of device to be used, but not a specific device. If you do not specify a device, the parameter will default to the disk or diskette having the volume serial number you specified on the VSN parameter.

BFSZ=n

Specifies the minimum I/O buffer size for the file you want to read from. Specifying a larger than minimum buffer size could reduce the number of disk I/O calls, causing your command to run faster. Not specifying a buffer size causes this parameter to default to a buffer size appropriate to the type of file you want to read from.

TRUNC= $\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$

Specifies whether input record truncation should be reported. If you omit this keyword or specify NO, EDT truncates records but displays no warning message.

KEY=start-col-no:end-col-no

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by the KEY parameter must contain decimal characters.

KKEY=start-col-no:end-col-no

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by the KKEY parameter must contain decimal characters.

SHOW∆first-col-no:last-col-no

Specifies the first and last column numbers of data to be used as input records. Data is read into the work-space file using the current line number and increment.

The purpose of the KEY and KKEY parameters is to make it convenient for you to update existing modules or files. These parameters do this by letting you correspond your work-space line numbers to sequence numbers in your work-space lines.

For example, let's say you have an existing file, named FILEA, containing these records:

```
01 NAME1 ADDRESS1
02 NAME2 ADDRESS2
03 NAME3 ADDRESS3
```

If you include KEY=1:2 on your READ command, EDT reads your file so that the records no longer contain the key field you specified on the KEY parameter. Instead, it assigns your key fields to be its work-space line numbers and places a decimal point in the fourth position to the left of the last column number.

In our example, EDT reads your file so that its work space looks like this:

```
.0001▷NAME1 ADDRESS1
.0002▷NAME2 ADDRESS2
.0003▷NAME3 ADDRESS3
```

Notice that EDT placed the decimal point four places to the left of column 2 and inserted zeros to hold the significant digits. Throughout your file, your work-space line numbers correspond to your sequence numbers in this way. So you'd always know how to reference a line. For example, to change NAME1 to NAMEA, you would know to specify:

```
@ON .0001 CHANGE 'NAME1' TO 'NAMEA'
```

The difference between the KEY and KKEY parameters is this: specifying KKEY causes EDT to retain the sequence numbers in your work-space line. For example (again using FILEA), if you include KKEY=1:2 on your READ command, EDT reads your file so your records still contain the columns specified with the KKEY parameter. But it assigns your key fields to be its work-space line numbers in the same way it did when you used KEY. Your work space now looks like this:

```
.0001▷01 NAME1 ADDRESS1
.0002▷02 NAME2 ADDRESS2
.0003▷03 NAME3 ADDRESS3
```

Notice that you would still use the same command to change NAME1 to NAMEA on line .0001. But, using KKEY instead of KEY on your READ command means that you can always see your original sequence numbers any time you do an @PRINT command.

The purpose of the SHOW parameter is to let you read into EDT's work space only specific columns of your existing file.

For example, let's assume you want to read your file, FILEA. But you want to read only the columns containing the names in the file. If you include SHOW 4:8 on your READ command, EDT reads in only the name field in columns 4 through 8 of the entire file. The work space looks like this:

```
1.ØØØØ▷NAME1
2.ØØØØ▷NAME2
3.ØØØØ▷NAME3
```

Now you can change the first name in the file by specifying:

```
@ON 1 CHANGE 'NAME1' TO 'NAMEA'
```

A typical example of reading a MIRAM data file is:

```
@R FIL=DATA1,VSN=RELS71,DEV=DISKE
```

This command reads a copy of the file DATA1 on diskette volume RELS71 into the current work-space file.

A typical example of sequentially reading a MIRAM data file by keys is:

```
@R FIL=DIVISIONS,VSN=RELØ7Ø,KEYN=1
```

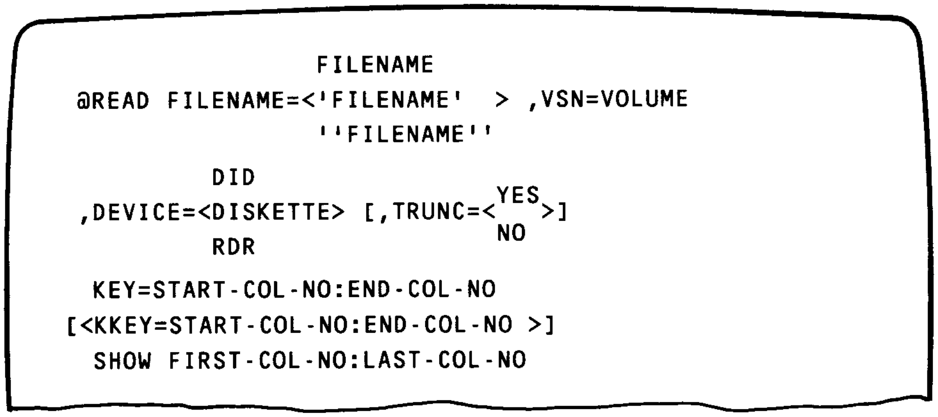
This command reads into the current work-space file a copy of the file DIVISIONS on disk volume REL070. Lines are sequentially read by the first key in every line.

Format 3

To read a unit record file from a data set label diskette or from the card reader, use this format:

```
@READ FILENAME=⎧FILENAME    ⎫,VSN=VOLUME
               ⎨'FILENAME'  ⎬
               ⎩''FILENAME''⎭
     ,DEVICE=⎧DID     ⎫⎡,TRUNC=⎧YES⎫⎤△⎡⎧KEY=START-COL-NO:END-COL-NO  ⎫⎤
             ⎨DISKETTE⎬⎣       ⎩NO ⎭⎦ ⎢⎨KKEY=start-col-no:end-col-no ⎬⎥
             ⎩RDR     ⎭               ⎣⎩SHOW△first-col-no:last-col-no⎭⎦
```

The help screen format is:



```
                    FILENAME
  @READ FILENAME=<'FILENAME'  > ,VSN=VOLUME
                    ''FILENAME''

          DID
                                    YES
  ,DEVICE=<DISKETTE> [,TRUNC=<         >]
                                    NO
          RDR

   KEY=START-COL-NO:END-COL-NO
 [<KKEY=START-COL-NO:END-COL-NO >]
   SHOW FIRST-COL-NO:LAST-COL-NO
```

where:

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the name of the file to be read. The file name is the LBL name of the file and can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces or parentheses are used in it. A file name is required only when using diskette.

```
VSN=volume
```

Specifies the volume serial number of the file. The number can be from one to six alphanumeric characters. This parameter is required for diskette unless you don't know the vsn of the diskette. In that case, you can use the DEVICE parameter instead. Neither the VSN nor the DEVICE parameter is required if you've catalogued the unit record file you want EDT to read into its work space.

```
DEVICE=⎧did     ⎫
       ⎨DISKETTE⎬
       ⎩RDR     ⎭
```

Specifies the device you want to read from. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISKETTE and RDR, specify the type of device to be used, but not a specific device. This parameter is required for the reader. If your unit record file is on diskette and you specified the VSN parameter, you do not have to specify the DEVICE parameter. Neither the VSN nor the DEVICE parameter is required if you've catalogued the unit record file you want EDT to read into its work space.

```
TRUNC={YES}
      {NO }
```

Specifies whether input record truncation should be reported. If you omit this keyword or specify NO, EDT truncates records but displays no warning message. This parameter applies only to MIRAM files.

```
KEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by KEY must contain decimal characters.

```
KKEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by KKEY must contain decimal characters.

```
SHOW△first-col-no:last-col-no
```

Specifies the first and last column numbers of data used as input records. Data is read into the work-space file using the current line number and increment.

NOTE:

For more information on the use of KEY, KKEY, and SHOW, see format 2 of the READ command.

A typical example of reading a unit record file is:

```
@R FIL="ACCOUNT FILE",VSN=USER05,DEV=DISKE
```

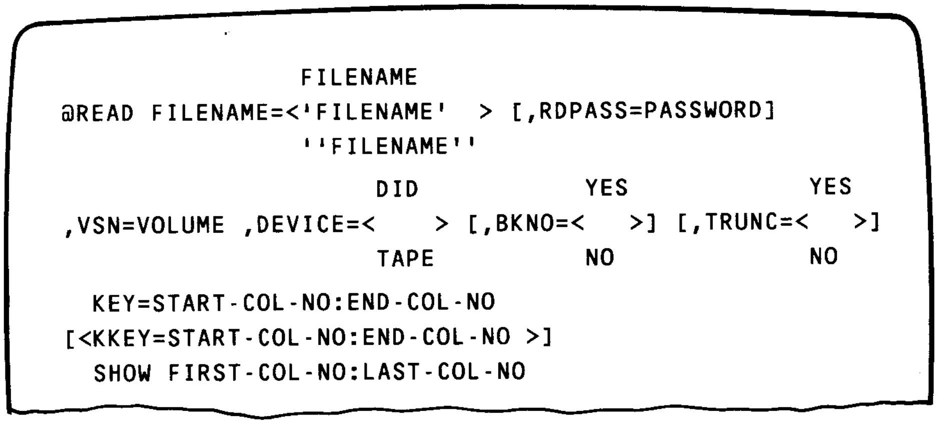
This command reads a copy of the file ACCOUNT FILE on diskette volume USER05 to the current work-space file.

Format 4

To read a file from a tape, use this format:

```
@READ FILENAME={filename    }[,RDPASS=password]
               {'filename'  }
               {''filename''}

   ,VSN=volume ,DEVICE={did }[,BKNO={YES}][,TRUNC={YES}]
                       {TAPE}[      {NO }][       {NO }]

    △[{KEY=start-col-no:end-col-no  }]
     [{KKEY=start-col-no:end-col-no }]
     [{SHOW△first-col-no:last-col-no}]
```

The help screen format is:



```
                  FILENAME
@READ FILENAME=<'FILENAME'  > [,RDPASS=PASSWORD]
                ''FILENAME''

                   DID          YES           YES
,VSN=VOLUME ,DEVICE=<    > [,BKNO=<   >] [,TRUNC=<   >]
                   TAPE         NO            NO

  KEY=START-COL-NO:END-COL-NO
[<KKEY=START-COL-NO:END-COL-NO >]
  SHOW FIRST-COL-NO:LAST-COL-NO
```

where:

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the name of the file to be read. The file name is the LBL name of the file and can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces or parentheses are used in it.

`RDPASS=password`

Specifies a read password that restricts access to the file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

`VSN=volume`

Specifies the volume serial number of the file. The number can be from one to six alphanumeric characters. You must specify this parameter unless it was previously entered in the system file catalog. If you do not know the vsn of the tape, you can specify the device address parameter (DEVICE) instead.

```
DEVICE=⎧did ⎫
       ⎩TAPE⎭
```

Specifies the device you want to read from. did specifies the device address (did) of the tape you want to use. The first digit is the channel number; the second and third specify the hardware address. TAPE specifies that you want to read from a tape; however, no specific tape is specified.

```
BKNO={YES}
     {NO }
```

Specifies whether or not the block numbers of the file you want to read should be checked. If you specify YES, the block numbers will be checked. If you specify NO, the block numbers will not be checked.

```
TRUNC={YES}
      {NO }
```

Specifies whether input record truncation should be reported. If you omit this keyword or specify NO, EDT truncates records but displays no warning message. This parameter applies only to MIRAM files.

```
KEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by the KEY parameter must contain decimal characters.

```
KKEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by the KKEY parameter must contain decimal characters.

```
SHOWΔfirst-col-no:last-col-no
```

Specifies the first and last column numbers of data to be used as input records. Data is read into the work-space file using the current line number and increment.

NOTE:

For more information on the use of KEY, KKEY, or SHOW, see format 2 of the READ command.

A typical example of reading a file from tape is:

```
@R FIL=NATIONAL,VSN=RELX71,DEV=TA
```

This command reads a copy of the file NATIONAL on tape volume RELX71 into the current work-space file.

Format 5

To read a file from the spool file to the current work-space file, use this format:

```
@READ [JOB=jobname] ⎡,HOLD= ⎧L⎫⎤
                    ⎢       ⎨N⎬⎥
                    ⎣       ⎩Y⎭⎦
     ⎡,FILENAME= ⎧filename    ⎫⎤ [,ACCT=acct-no]
     ⎢           ⎨'filename'  ⎬⎥
     ⎣           ⎩''filename''⎭⎦
     ,QUEUE= ⎧LOG  ⎫ ⎡,ALL= ⎧YES⎫⎤ ⎡,SKIP= ⎧n⎫⎤ ⎡,TRUNC= ⎧YES⎫⎤
             ⎪PRINT⎪ ⎣      ⎩NO ⎭⎦ ⎣       ⎩0⎭⎦ ⎣        ⎩NO ⎭⎦
             ⎪PUNCH⎪
             ⎩RDR  ⎭
     Δ⎡⎧KEY=start-col-no:end-col-no   ⎫⎤
      ⎢⎨KKEY=start-col-no:end-col-no  ⎬⎥
      ⎣⎩SHOWΔfirst-col-no:last-col-no ⎭⎦
```

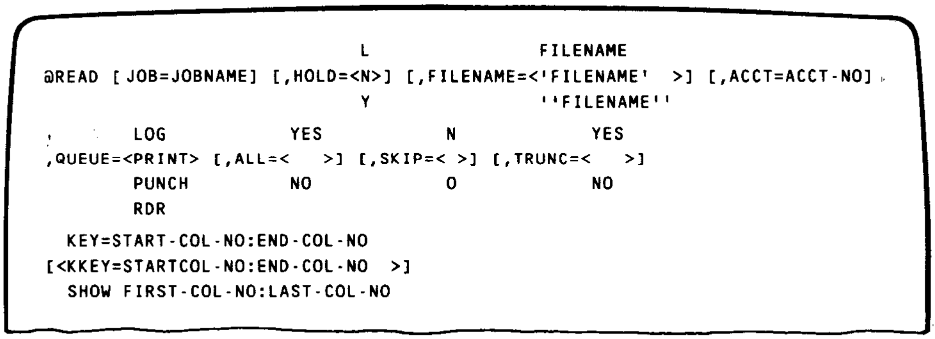
NOTES:

1. *In order to read a spool file that was created at a workstation, you must issue the @READ command under the same user-id that was in effect when the spool file was created. You can enter the user-id via the LOGON command or by using the JCL option statement. This requirement protects spool files from unauthorized access and unintentional destruction.*

2. *When you create spool files at the system console in console mode, you can later access these files only in console mode. When you create spool files at the system console in workstation mode, you can later access them only by using the same user-id you used when creating them.*

3. *If you use the file name, it must be either the LBL name or the job name concatenated with the LFD name of the program that created the file.*

The help screen format is:



```
                                      L                 FILENAME
@READ [JOB=JOBNAME] [,HOLD=<N>] [,FILENAME=<'FILENAME'  >] [,ACCT=ACCT-NO]
                                      Y                 ''FILENAME''
        LOG          YES          N            YES
,QUEUE=<PRINT> [,ALL=<    >] [,SKIP=< >] [,TRUNC=<    >]
        PUNCH        NO           0            NO
        RDR
  KEY=START-COL-NO:END-COL-NO
[<KKEY=STARTCOL-NO:END-COL-NO  >]
  SHOW FIRST-COL-NO:LAST-COL-NO
```

where:

```
JOB=jobname
```

Specifies the name of the job that produced the spool file.

```
HOLD=⎧L⎫
     ⎨N⎬
     ⎩Y⎭
```

Specifies whether or not a spooled file has been placed in a HOLD state. Files are held in two ways. The first is through the use of a HOLD command, from the workstation or system console, or the inclusion of a HOLD job control parameter before processing. The second is by specifying the file to be retained after processing. A file placed in the HOLD state by the first method has not been processed. A file placed in the HOLD state by the second method has been processed at least once.

If the file you want to read is a log file, and is in the HOLD state, enter L. If the file you want to read is a type other than a log file and is in a HOLD state, enter Y. If the file you want to read, log or other type, is not in a HOLD state, enter N. If the file you want to read is in the RDR spool queue and was specified to be retained after processing, it will not be in a HOLD state. Retained input files are not placed on HOLD.

If you omit the HOLD parameter and the file you want to read isn't on the reader queue, EDT locates the file regardless of what you specified for the HOLD spooling command for that file.

```
FILENAME=⎧filename      ⎫
         ⎨'filename'    ⎬
         ⎩''filename''  ⎭
```

Specifies the logical name of the spool file and must be the LBL name from your job control or the job name concatenated with the LFD name. It can be from 1 to 17 alphanumeric characters. The file name must be enclosed in apostrophes when embedded spaces, parentheses, or commas are used in it.

```
ACCT=acct-no
```

Specifies the job account number that created the spool file. The account number may be from one to four alphanumeric characters. It's always optional.

```
QUEUE=⎧LOG  ⎫
      ⎪PRINT⎪
      ⎨PUNCH⎬
      ⎩RDR  ⎭
```

Indicates the spool queue containing the file you want to read. You may enter LOG, PRINT, PUNCH, or RDR. LOG indicates that the file is in the log queue. PRINT indicates that the file is in the printer queue. PUNCH indicates that the file is in the card punch queue. RDR indicates that the file is in the card reader queue. Specifying QUEUE=RDR forces the HOLD parameter to NO.

```
ALL=⎧YES⎫
    ⎩NO ⎭
```

Specifies whether or not all spool files with the specified parameters are to be read. If you specify YES, all the spool subfiles with the specified parameters are read. If you specify NO, only the first spool file found with the specified parameters is read.

```
SKIP=⎧n⎫
     ⎩0⎭
```

Specifies the number of spool files, with parameter values identical to those specified, that are to be skipped before reading a file. The default value 0 indicates that the first file found matching the specified parameters is read. n is any integer.

```
TRUNC=⎧YES⎫
      ⎩NO ⎭
```

Specifies whether input record truncation should be reported. If you omit this keyword or specify NO, EDT truncates records but displays no warning message. This parameter applies only to MIRAM files.

```
KEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by the KEY parameter must contain decimal characters.

```
KKEY=start-col-no:end-col-no
```

Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by the KKEY parameter must contain decimal characters.

```
SHOWΔfirst-col-no:last-col-no
```

Specifies the first and last column numbers of data to be used as input records. Data is read into the work-space file using the current line number and increment.

NOTE:

For more information on the use OF KEY, KKEY, or SHOW, see format 2 of the READ command.

A typical example of reading a file from the spool file is:

```
@R FIL=PAYFILE,Q=PR
```

This command reads into the current work-space file a copy of the file PAYFILE from the printer queue of the spool file.

Format 6

To read the same module or file last accessed through a previous @READ or @WRITE command into EDT's work space, use this format:

```
@READ
```

NOTE:

There is no help screen for format 6 of the READ command.

For example, let's say you create a library module and write it to a disk using the command, @WRITE MO=USER01,FIL=FILE1,VSN=REL080,SI=1,SAT=YES. After you write it out to the disk, however, you realize you need to make another change to it. To read the file back into EDT's work space, you simply enter @READ and EDT reads your module back into its work space.

Format 7

To read the same module or file last accessed through a previous @READ or @WRITE command but read now with a KEY, KKEY, or SHOW parameter or any valid EDT command specified, use this format:

```
@READΔ;Δ⎡⎧KEY=start-col-no:end-col-no    ⎫⎤[valid EDT command]
        ⎢⎨KKEY=start-col-no:end-col-no   ⎬⎥
        ⎣⎩SHOWΔfirst-col-no:last-col-no⎭⎦
```

NOTE:

There is no help screen for format 7 of the READ command.

where:

;
: Specifies the same file parameters, with the exception of the KEY, KKEY, and SHOW, used in the most recent @READ or @WRITE command.

KEY=start-col-no:end-col-no
: Specifies the starting and ending column number for keys used to read the file. The keys are used as the current work-space line numbers. They are deleted from their locations (as data) in the lines, and the lines are condensed accordingly. The columns specified by the KEY parameter must contain decimal characters.

KKEY=start-col-no:end-col-no
: Specifies the starting and ending column numbers for keys used to read the file. The keys are used as the current work-space line numbers and are also retained as data in the lines. The columns specified by the KKEY parameter must contain decimal characters.

SHOWΔfirst-col-no:last-col-no
: Specifies the first and last column numbers of data to be used as input records. Data is read into the work-space file using the current line number and increment.

NOTE:

For more information on the use of KEY, KKEY, and SHOW, see format 2 of the READ command.

valid EDT command
: Specifies any EDT command that you want to combine with the READ command. Table 3–2 lists the valid commands you can combine with the READ command.

Here's one example of the semicolon format:

Let's say you have an existing file in EDT's work space that looks like this:

```
Ø1 NAME1 ADDRESS1 PHONE-NO1 ACCOUNT-NO1
Ø2 NAME2 ADDRESS2 PHONE-NO2 ACCOUNT-NO2
Ø3 NAME3 ADDRESS3 PHONE-NO3 ACCOUNT-NO3
```

You want to create a subset of this file in a new file so you include the SHOW parameter on your @WRITE command, like this:

```
@WRITE FIL=MYFILE,VSN=RELØ8Ø,SI=1 SHOW 1:26
```

This command writes only the sequence number, name, address, and phone number fields to your new file. Next, you want to read your new file back into EDT's work space, but you want to ignore the SHOW parameter so you can specify a KEY parameter, like this:

```
@READ ; KEY=1:2
```

The semicolon tells EDT to read in the same file in the last @WRITE command but to ignore the SHOW parameter because you want to specify a KEY parameter. The KEY parameter tells EDT to read your file so that EDT's work-space line numbers correspond to the sequence numbers in your file. For more information on KEY, KKEY, and SHOW, see format 2 of the READ command.

Another example of the semicolon format is:

```
@READ ; change 'ABC' to 'DEF'
```

This command reads into EDT's work space the file specified in the previous READ or WRITE command but changes all occurrences of ABC to DEF.

Some typical examples of the READ command combined with other commands are:

@R FIL='FILE H',VSN=D00029 SEQ '1000' BY 5

Reads a copy of the file 'FILE H' on disk volume D00029 into the work-space file and inserts sequence numbers in columns 1 through 4 on each line, beginning with the sequence string 1000 and incrementing it by 5. Note that BY 5 is not a parameter associated with the READ command, but is an EDT modifier, which we explain in Section 4.

@R FIL=MYFILE,RD=LT05,VSN=USER01 FIN 'JOHN'

Reads a copy of the file MYFILE (with password LT05) on disk volume USER01 into the work-space file and locates the first occurrence of JOHN (assigning the corresponding line and first and last column numbers to ?, [, and], respectively).

REM

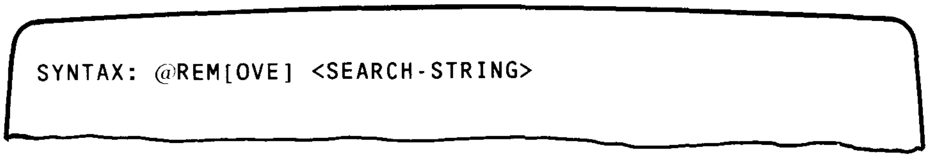
3.15. DELETING DATA (@REMOVE)

The REMOVE command deletes a specified string from lines in the work-space file.

The format is:

```
@REMOVE 'search-string'[*n]
```

The help screen format is:



```
SYNTAX: @REM[OVE] <SEARCH-STRING>
```

where:

'search-string'
: Specifies a string that is to be deleted from the work-space file. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

The REMOVE command used in this basic format deletes the first occurrence of the specified string on each line in the work-space file. Lines are condensed to adjust for the deleted string.

Some typical examples of the REMOVE command are:

@REM 'TOM'
Deletes the first occurrence of TOM on every line in the current work-space file. If TOM appears more than once on a line, the remaining occurrences are not deleted. This command is equivalent to @CHANGE 'TOM' TO '' .

@REM '.'*10
Deletes the first occurrence of .... (a separator between data fields, for example) on every line in the current work-space file

The following command modifiers can be added to the REMOVE command, increasing its capability: ALL, COLUMN, FIRST, ON.

Commands that can be combined with the REMOVE command that permit faster and more efficient editing are: COPY, DELETE, FIND, FSTATUS, LIST, MOVE, PRINT, PUNCH, READ, UPDATE, and WRITE.

Some typical REMOVE examples using modifiers and other commands are:

```
@REM ALL 'BOOKS'
```

Deletes all occurrences of BOOKS in the current work-space file

```
@REM 'TEAM #3' PRINT
```

Deletes the first occurrence of TEAM #3 on every line in the current work-space file and displays the lines where these changes were made

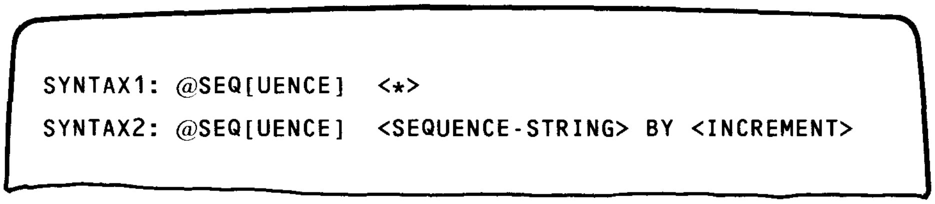
SEQ

3.16. PLACING SEQUENCE NUMBERS INTO EXISTING LINES (@SEQUENCE)

The SEQUENCE command places sequence numbers into existing lines in the current work-space file. The SEQUENCE command lets you resequence data. The format for the SEQUENCE command is:

```
@SEQUENCE{'sequence-string'[*n] BY increment}
         {*                                 }
```

The help screen format is:



```
SYNTAX1: @SEQ[UENCE]  <*>
SYNTAX2: @SEQ[UENCE]  <SEQUENCE-STRING> BY <INCREMENT>
```

where:

'sequence-string'
: Specifies an alphanumeric string that is to be inserted on an existing line in the work-space file. The sequence string must be enclosed in apostrophes and one or more of its rightmost characters must be an integer from 0 to 9. A sequence string of up to 15 decimal digits may be specified. Multiple copies of the sequence string may be specified by using the string multiplication factor *n, where n is the number of times the original sequence string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

increment
: Sets the increment for sequence strings inserted into existing lines in the work-space file. A number of up to nine decimal digits may be specified.

*
: Indicates that the sequence string to be entered is the current work-space line number.

The sequence string used in this format places sequence numbers on each line, beginning in column 1. When sequence strings are inserted into existing lines, they replace the contents of the columns rather than shifting the contents of the line to incorporate the string.

Some typical examples of the sequence command are:

```
@SEQ 'ED20'*2 BY 5
```

Overlays columns 1 through 8 on the first line in the current work-space file with the sequence string D20ED20 and overlays columns 1 through 8 on subsequent (existing) lines with increments of 5 of that string

```
@SEQ 'A100' BY 10
```

Overlays columns 1 through 4 on the first line of the current work-space file with the sequence string A100 and overlays columns 1 through 4 on subsequent (existing) lines with increments of 10 of that string

Command modifiers that can be added to the SEQUENCE command are ON and COLUMN. ON allows you to insert sequence numbers on specific lines rather than on all lines in the current work-space file; COLUMN allows you to insert sequence numbers beginning in a column other than column 1.

Commands that can be combined with the SEQUENCE command that permit faster and more efficient editing are: COPY, DELETE, FIND, FSTATUS, LIST, MOVE, PRINT, PUNCH, READ, UPDATE, and WRITE.

Some typical SEQUENCE examples using modifiers and other commands are:

```
@COL 73 SEQ '00001000' BY 1
```

Inserts sequence numbers in columns 73 through 80 on all existing lines in the current work-space file

```
@WRITE FIL=MYFILE,VSN=D00027 SEQ *
```

Inserts sequence numbers into existing lines while writing the work-space file to a permanent file. Sequence numbers are inserted in columns 1 through 8 on every line beginning with the current work-space line number (the special symbol *) as the first sequence string.

U

3.17. UPDATING LINES (@UPDATE)

The UPDATE command displays specified lines from the work-space file one at a time for you to edit or change. You can update a single line, nonconsecutive lines, consecutive lines, lines containing a specified string, or any combination of these lines.

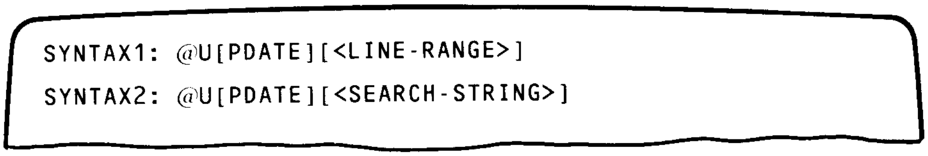
The format is:

```
@UPDATE [line-range]['search-string'[*n]]
```

NOTE:

You can't use the @UPDATE command in screen mode. Instead, use the @ROLL screen command (Section 8).

The help screen format is:



```
SYNTAX1: @U[PDATE][<LINE-RANGE>]
SYNTAX2: @U[PDATE][<SEARCH-STRING>]
```

where:

`line-range`
Specifies by line numbers, special symbols, or EDT variables a single line, nonconsecutive lines, consecutive lines, or any combination of these lines to be updated. Consecutive line ranges must be specified in ascending order.

`'search-string'`
Specifies by a string in the lines those lines to be updated. The search string is usually enclosed in apostrophes; however, it may also be enclosed in quotes or a combination of quotes and apostrophes. Only one search string may be specified. Multiple copies of the search string may be specified by using the string multiplication factor *n, where n is the number of times the original search string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

If you do not specify a line range or a search string for the @UPDATE command, EDT displays every line in the EDT work space one line at a time so you can update them.

Once you enter the UPDATE command and EDT displays a data line, you may update the data line in five different ways. You can enter:

- @STRIKE to delete the entire line;
- @NOCHANGE to leave the line unchanged;
- @CONTINUE to continue the UPDATE command without displaying the lines;
- an EDT command to terminate the entire UPDATE command and execute the specified command; or
- a new data line to replace the existing line.

NOTE:

The tab character is ignored during an @UPDATE process.

You may key in these entries in two different locations on the screen. You may enter them at the end of the displayed line, but you must precede the entry with a start-of-entry (SOE) character (△). Or, you may move the cursor back to the beginning of the line and key in the entry over the existing line.

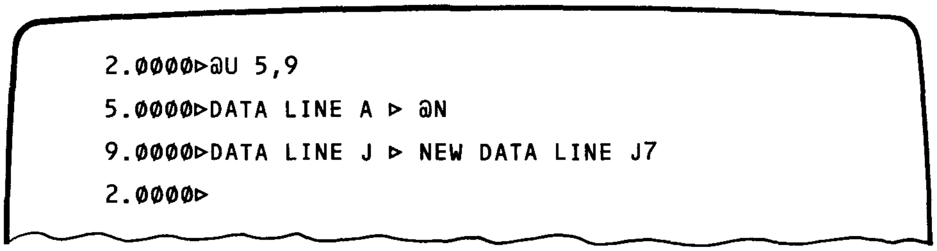
NOTES:

1. *If a command is to be interpreted as a data line, remember to key in a double at sign (@@) rather than a single at sign (@).*
2. *The STRIKE and NOCHANGE options cannot be used if the UPDATE command is combined with the COPY command.*
3. *A null line in response to the UPDATE command deletes the line, that is, pressing the XMIT key with the cursor at the start-of-entry field.*

Some typical examples of the UPDATE command are:

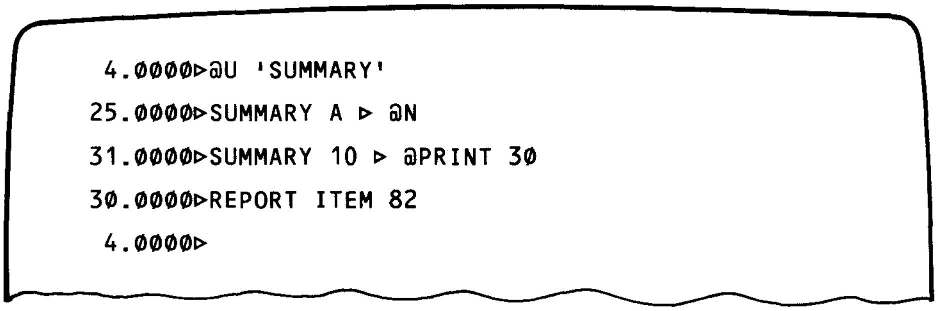
```
 1.ØØØØ▷@U 4Ø
4Ø.ØØØØ▷DATA LINE TO BE UPDATED. ▷ @ST
 1.ØØØØ▷
```

Here, the @U 40 command on line 1 displays line 40 to be updated. We delete the line by entering ▷ @ST at the end of the line and then transmitting it.



```
2.0000▷@U 5,9
5.0000▷DATA LINE A ▷ @N
9.0000▷DATA LINE J ▷ NEW DATA LINE J7
2.0000▷
```

Here, the @U 5,9 command on line 2 displays lines 5 and 9 one at a time to be updated. After each line is displayed, it is updated. We leave line 5 unchanged by entering ▷ @N at the end of line 5 and then transmitting it; we replace line 9 with a new data line by entering an SOE (▷) character and the new data at the end of line 9 and transmitting it.



```
 4.0000▷@U 'SUMMARY'
25.0000▷SUMMARY A ▷ @N
31.0000▷SUMMARY 10 ▷ @PRINT 30
30.0000▷REPORT ITEM 82
 4.0000▷
```

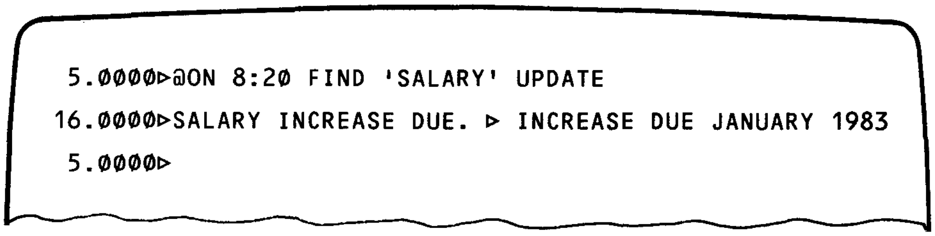
Here, the @U 'SUMMARY' command on line 4 displays all lines containing SUMMARY in the current work-space file one at a time so that they can be updated. Notice that we leave line 25 (the first line in the work-space file containing SUMMARY) unchanged by entering ▷ @N at the end of the line and transmitting it. We then terminate the entire UPDATE command at line 31 (the second line containing SUMMARY) by entering an SOE character (▷) and the EDT command @PRINT 30 at the end of line 31 and transmitting it.

Lines are updated in the order that they are specified in the command. When using a search string, lines are updated in the order that they appear in the work-space file.

The following command modifiers can be added to the UPDATE command, altering its capability: ALL, COLUMN, FIRST, and ON.

Commands that may be combined with the UPDATE command that permit faster and more efficient editing are: CHANGE, COPY, DELETE, FIND, INSERT, LIST, MOVE, NUMBER, PRINT, REMOVE, and SEQUENCE.

A typical example of the UPDATE command used with modifiers and other commands is:



```
 5.ØØØØ▷@ON 8:2Ø FIND 'SALARY' UPDATE
16.ØØØØ▷SALARY INCREASE DUE. ▷ INCREASE DUE JANUARY 1983
 5.ØØØØ▷
```

Here, the command on line 5 locates the first occurrence of SALARY on lines 8 through 20 and displays the line for updating. The user keys in ▷ INCREASE DUE JANUARY 1983. EDT makes the change in the work space but doesn't display the changed line because the user didn't enter an @PRINT command. EDT displays line 5.000 to accept the next command.

W

3.18. CREATING AND SAVING DATA (@WRITE)

The WRITE command writes a copy of the current work-space file to a program library module or data file on disk, diskette, tape, or to the spool file. If the file being written to does not already exist, the WRITE command enables you to create it and then write to it. If the file already exists, it can be overwritten with a copy of the current work-space file, or a copy of the current work-space file can be appended to the end of the file. The WRITE command can also copy the current work-space file to cards or the printer.

The WRITE command is divided into seven basic formats. The formats are used to write the current work-space file to:

a SAT or MIRAM library module on a disk or format label diskette;

a MIRAM data file on a disk or format label diskette;

a unit record file to a printer, card punch, or a data set label diskette;

a file to a tape;

a file to the spool file;

the same module or file last accessed through a previous @READ or @WRITE command; or

the same module or file last accessed through a previous @READ or @WRITE command but written to now with any valid EDT command specified.

The following paragraphs describe these five formats and give examples for each. Here is some general information about the WRITE command that pertains to all five formats:

- Note the following information about how your system accesses files before writing them from the EDT work space to their permanent files. OS/3 does the following:
  - permits variable or fixed record length (1 to 2048 characters, depending on your specification for the BUFFER parameter of the @SET directive);
  - permits duplicate keys on multiple indexed file types;
  - assumes no change on index fields on multiple indexed file types, but allows you to change them if you want to; and
  - can't compile, link, or access programs as MIRAM program modules. If you want your system to compile, link, or access a program, it must be written as a module of a SAT file.

- You can enter the WRITE command via keywords or positional parameters. We discuss keyword version here because it's used most often. For information on how you can enter the WRITE command via positional parameters, see Appendix E.

- Unlike the keywords in the other EDT commands, the keywords in the WRITE command can't be separated by spaces. If you leave spaces between them, EDT can't process the command and you'll receive an error message.

- Commands that you can combine with the WRITE command for more efficient and faster editing are: CHANGE, COPY, FIND, INSERT, REMOVE, and SEQUENCE. We provide examples of the @WRITE command in combination with other commands after the descriptions of the five formats so that you're familiar with the various parameters associated with the WRITE command when studying the examples.

- As we explained in the statement of conventions, we show the help screen formats you'll see when you issue an @PROMPT screen command along with the formats we present for the @WRITE command. You should note that you will not immediately see the help screens for the seven formats of the @READ command when you first issue the @PROMPT command to get help with the @WRITE command. To see help screens for the seven formats, you must follow the procedure for getting additional help from an @PROMPT command help screen. That is, you must press F13 for additional help until you see the format for the type of file you want to write to. We describe the procedure for getting additional help from an @PROMPT command help screen in full detail in Section 8.

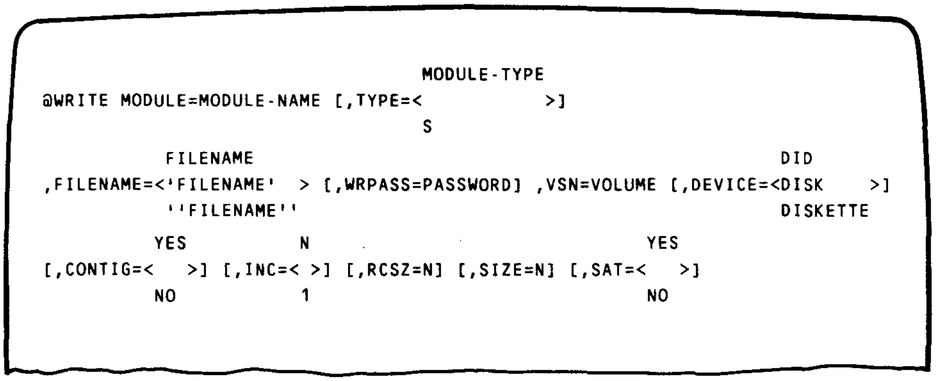
- After you issue an @WRITE command, issue an @DELETE command if you're not finished your EDT session. The @DELETE command erases lines in EDT's work space so that the lines you already wrote to a permanent file aren't included with the lines you intend to enter next.

Format 1

To write the current work-space file to a SAT or MIRAM library module on a disk or format label diskette, use this format:

```
@WRITE MODULE=module-name ⎡,TYPE=⎧module-type⎫⎤
                          ⎣      ⎩S          ⎭⎦

     ,FILENAME=⎧filename    ⎫[,WRPASS=password],VSN=volume⎡,DEVICE=⎧did     ⎫⎤
               ⎨'filename'  ⎬                             ⎢        ⎨DISK    ⎬⎥
               ⎩''filename''⎭                             ⎣        ⎩DISKETTE⎭⎦

     ⎡,CONTIG=⎧YES⎫⎤⎡,INC=⎧n⎫⎤[,RCSZ=n][,SIZE=n]
     ⎣        ⎩NO ⎭⎦⎣     ⎩1⎭⎦

     ⎡,SAT=⎧YES⎫⎤
     ⎣     ⎩NO ⎭⎦
```

The help screen format is:



```
                                   MODULE-TYPE
@WRITE MODULE=MODULE-NAME [,TYPE=<            >]
                                   S

      FILENAME                                             DID
,FILENAME=<'FILENAME'  > [,WRPASS=PASSWORD] ,VSN=VOLUME [,DEVICE=<DISK    >]
      ''FILENAME''                                         DISKETTE

     YES         N                          YES
[,CONTIG=<   >] [,INC=< >] [,RCSZ=N] [,SIZE=N] [,SAT=<   >]
     NO          1                          NO
```

where:

MODULE=module-name

Specifies the name of the library module to be written. The module can be a language source module or a job control stream. If you're writing a job control stream, use its job name as the module name. In any case, the module name may be an existing name or one you create. It can be from one to eight alphanumeric characters with the first character being alphabetic.

TYPE={module-name
S}

Specifies the type of module to be written. For SAT files, the types permitted are source (S), macro (M), procedure (P), load (L), object (O), or proc-name (PN). For MIRAM files, you may specify screen format (F or FC), help screen (HELP), saved run library (J), or menu (MENU).

You may create your own module types, identifying them with a 1- to 4-character type. The module types can serve as qualifiers for the modules they are associated with.

FILENAME={filename
'filename'
''filename''}

Specifies the name of the file where the module is to be written. The file name is the LBL name of the file and can be an existing LBL name, or, if you are creating the module now, you supply the LBL name with this parameter now. It can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, commas, or parentheses are used in it.

WRPASS=password

Specifies the write password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

VSN=volume

Specifies the volume serial number of the file where the module is to be written. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously entered the file in the system file catalog. If you do not know the vsn of the disk or diskette you want to write to, you can specify the DEVICE parameter instead.

$\underline{\text{DEV}}\text{ICE=}\begin{Bmatrix}\text{did}\\ \text{DISK}\\ \text{DISKETTE}\end{Bmatrix}$

Specifies the device you want to write to. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and the third specify the hardware address. The other choices, DISK and DISKETTE, specify the type of device you want to write to, but not the specific device. If you do not specify a device, the parameter will default to the disk or diskette having the volume serial number you specified on the VSN parameter.

$\underline{\text{CON}}\text{TIG=}\begin{Bmatrix}\text{YES}\\ \underline{\text{N}}\text{O}\end{Bmatrix}$

Specifies whether or not EDT allocates contiguous storage space to the file. If you specify YES, contiguous storage space is allocated. If you specify NO, noncontiguous storage space is allocated.

$\text{INC=}\begin{Bmatrix}\text{n}\\ \text{1}\end{Bmatrix}$

Specifies the number of cylinders to be added to a file each time it requires more storage space. This parameter is specified only when you create a new file. The default is 1.

$\underline{\text{RCS}}\text{Z=n}$

Specifies the record size used when writing a MIRAM library module to a disk or format label diskette. The default is 256 bytes. Since EDT determines the record size of a SAT module according to the module type, this parameter applies only to MIRAM library modules.

$\underline{\text{SI}}\text{ZE=n}$

Specifies the number of cylinders initially allocated for a new file. This parameter is specified only when you create a new MIRAM file.

$\underline{\text{SA}}\text{T=}\begin{Bmatrix}\underline{\text{Y}}\text{ES}\\ \underline{\text{N}}\text{O}\end{Bmatrix}$

Specifies whether the output file is a SAT or a MIRAM file. If the keyword is omitted, a MIRAM file is assumed. MIRAM files are used for data, screen format services modules, and saved job control modules, while SAT files are used for library modules. This parameter is required only when you are creating a new file.

A typical example of writing a library module to disk is:

```
@W MO=MYPROG,FIL=LIBFILE,VSN=DØØØ24
```

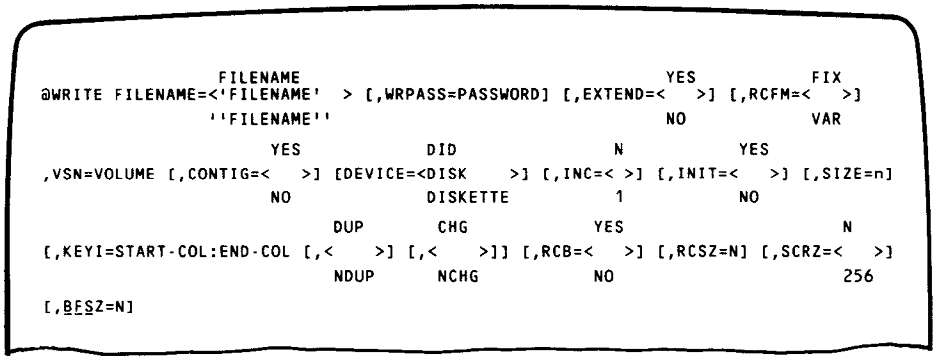
This command writes the source module MYPROG to the existing SAT file LIBFILE on disk volume D00024.

Format 2

To write the current work-space file to a MIRAM data file on a disk or format label diskette, use this format:

```
@WRITE FILENAME={filename    }[,WRPASS=password]
                {'filename'  }
                {''filename''}
   ,VSN=volume [,CONTIG={YES}] [,DEVICE={did     }] [,INC={n}] [,INIT={YES}]
                        {NO }           {DISK    }         {1}         {NO }
                                        {DISKETTE}
   [,EXTEND={YES}]
            {NO }
   [,KEYi={start-col-no:end-col-no                      }]
          {(start-col-no:end-col-no,{DUP },{CHG })      }
                                    {NDUP} {NCHG}
   ,SIZE=n [,RCB={YES}] [,RCFM={FIX}]
                 {NO }         {VAR}
   ,RCSZ=n [,SCSZ={n  }] [,BFSZ=n]
                  {256}
```

The help screen format is:



```
                 FILENAME                                          YES          FIX
@WRITE FILENAME=<'FILENAME'  > [,WRPASS=PASSWORD] [,EXTEND=<    >] [,RCFM=<   >]
                ''FILENAME''                                       NO           VAR

                      YES             DID                 N            YES
,VSN=VOLUME [,CONTIG=<    >] [DEVICE=<DISK     >] [,INC=< >] [,INIT=<    >] [,SIZE=n]
                      NO              DISKETTE            1            NO

                               DUP        CHG             YES                      N
[,KEYI=START-COL:END-COL [,<     >] [,<     >]] [,RCB=<    >] [,RCSZ=N] [,SCRZ=<   >]
                               NDUP       NCHG            NO                       256

[,BFSZ=N]
```

where:

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the name of the file to be written. The file name is the LBL name of the file and can be an existing LBL name, or, if you are creating the file now, you supply the LBL name with this parameter now. The file name can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes whenever embedded spaces, commas, or parentheses are used in it.

```
WRPASS=password
```

Specifies a write password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

```
VSN=volume
```

Specifies the volume serial number of the disk or diskette where the specified file is to be written. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously specified the file in the system file catalog. If you do not know the vsn of the disk or diskette you want to write to, you can specify the DEVICE parameter instead.

```
CONTIG=⎧YES⎫
       ⎩NO ⎭
```

Specifies whether or not EDT allocates contiguous storage space to the file. If you specify YES, contiguous storage space is allocated. If you specify NO, noncontiguous storage space is allocated.

```
DEVICE=⎧did     ⎫
       ⎨DISK    ⎬
       ⎩DISKETTE⎭
```

Specifies the device you want to write to. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISK and DISKETTE, specify the type of device to be used, but not a specific device. If you do not specify a device, the parameter will default to the disk or diskette having the volume serial number you specified on the VSN parameter.

```
INIT={YES}
     {NO }
```

Specifies whether the contents of a file are to be overwritten with new data being added. If you specify NO (the default), the old information remains intact and the new data is added to the end of the file. If you specify YES, the original file contents are overwritten with the new data, but the original file characteristics are not preserved. (The only exception would be if the original file had been created using the default values for all the file parameters.) INIT assumes the default values for all file parameters. To overwrite a file using the old file characteristics, use EXTEND=NO, not INIT. Otherwise, use INIT and specify your new file parameters. INIT applies only to MIRAM data files and tape files.

```
EXTEND={YES}
       {NO }
```

Specifies whether the contents of a file are to be overwritten or merged with new data. If you take the default, YES, the existing file remains intact and the new data is added to the end. If you specify NO, the data is overwritten, but the file characteristics are preserved. To overwrite a file using file characteristics, use the INIT=YES parameter instead of EXTEND. (The default values for EXTEND and INIT specify the same condition. The two parameters differ in how a file is overwritten with new data.) The EXTEND parameter is ignored if you specify INIT=YES. EXTEND applies only to MIRAM data files and tape files.

```
INC={n}
    {1}
```

Specifies the number of cylinders to be added to a file each time it requires more storage space. This parameter is specified only when you create a new file. The default is 1.

```
KEYi={start-col-no:end-col-no                                     }
     {(start-col-no:end-col-no,{DUP },{CHG })}
     {(                        {NDUP} {NCHG})}
```

Specifies the starting (start-col-no) and ending (end-col-no) column positions of a key (i=1–5). You may repeat this parameter for as many of the five keys as you have available to you when creating a file. DUP/NDUP specifies whether or not duplicate keys are allowed. CHG/NCHG specifies whether or not keys may be changed.

```
SIZE=n
```

Specifies the number of cylinders initially allocated for a file. You specify this parameter only when creating a new MIRAM file.

```
RCB={YES}
    {NO }
```

Specifies whether or not a MIRAM file should be created with a record control byte. If you specify YES, the file is created with a record control byte. If you specify NO or omit the keyword, it is not.

$\underline{\text{RCF}}\text{M=}\begin{Bmatrix}\boxed{\text{FIX}}\\ \text{VAR}\end{Bmatrix}$

Specifies the record format of the file. If you want to write variable-length records, enter VAR. If you want to write fixed-length records, omit the keyword or enter FIX.

$\underline{\text{RCS}}\text{Z=n}$

Specifies the record size used when writing a MIRAM data file to a disk or a format label diskette. The default is 256.

$\underline{\text{SCS}}\text{Z=}\begin{Bmatrix}\text{n}\\ \boxed{256}\end{Bmatrix}$

Specifies the sector size of the disk pack when creating a MIRAM file. Use this parameter only when writing to a file on a volume mounted on a selector channel device. The following are selector channel devices:

8414 disk drive

8425 disk drive

8430 disk drive

8433 disk drive

NOTE:

These devices are available only on Series 90 hardware.

$\underline{\text{BFS}}\text{Z=n}$

Specifies the minimum I/O buffer size for the file you want to write to. Specifying a larger than minimum buffer size could reduce the number of disk I/O calls, causing your command to run faster. Not specifying a buffer size causes this parameter to default to a buffer size appropriate to the type of file you are writing to.

A typical example of writing a MIRAM file to disk is:

```
@W FIL=LISTFILE,WR=LPM,VSN=D00029,KEY1=50:58
```

This command writes the current work-space file to the file LISTFILE on disk volume D00029. The password LPM is supplied because the file was previously cataloged with it. Note that the file is written using the first key (i.e., the key specified in columns 50 through 58).

NOTE:

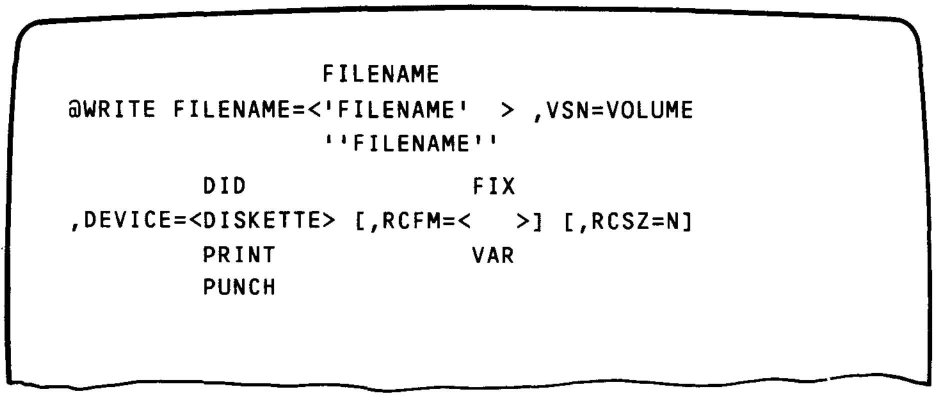
When EDT writes records from its work space, it strips them of trailing spaces. Therefore, whenever you're writing MIRAM data files with keys in the last columns of their records, be sure to do one of the following actions to prevent future data management errors: (1) don't use keys that end in space, or (2) specify the STRIP=OFF parameter on the @SET directive so EDT won't strip records of trailing spaces.

Format 3

To write the current work-space file to a unit record file (i.e., to the printer, card punch, or to a data set label diskette), use this format:

```
@WRITE FILENAME=⎧filename     ⎫,VSN=volume
                ⎨'filename'   ⎬
                ⎩''filename'' ⎭
     ,DEVICE=⎧did      ⎫⎡,RCFM=⎧FIX⎫⎤[,RCSZ=n]
             ⎪DISKETTE ⎪⎣      ⎩VAR⎭⎦
             ⎨PRINT    ⎬
             ⎩PUNCH    ⎭
```

The help screen format is:



```
                     FILENAME
@WRITE FILENAME=<'FILENAME'  > ,VSN=VOLUME
                     ''FILENAME''

          DID               FIX
,DEVICE=<DISKETTE> [,RCFM=<    >] [,RCSZ=N]
          PRINT             VAR
          PUNCH
```

where:

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the name of the file to be written. The filename is the LBL name of the file and can be an existing LBL name, or, if you are creating the file now, you supply the LBL name with this parameter now. It can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, commas, or parentheses are used in it.

```
VSN=volume
```

Specifies the volume serial number of the media where the specified file is to be written. The number can be from one to six alphanumeric characters. This parameter is required for diskette unless you don't know the vsn of the diskette. In that case, you can use the DEVICE parameter instead. Neither the VSN nor the DEVICE parameter is required if you've cataloged the unit record file you want EDT to write its work-space contents to.

```
DEVICE=⎧did     ⎫
       ⎪DISKETTE⎪
       ⎨PRINT   ⎬
       ⎩PUNCH   ⎭
```

Specifies the device you want to write to. did specifies the device address (did) of the device you want to use. The first digit is the channel number; the second and third specify the hardware address. The other choices, DISKETTE, PRINT, and PUNCH, specify the type of device to be used, but not a specific device. This parameter is required for the printer or the punch. If your unit record file is on diskette and you specified the VSN parameter, you do not have to specify the DEVICE parameter. Neither the VSN nor the DEVICE parameter is required if you've catalogued the unit record file you want EDT to write its work-space contents to.

```
RCFM=⎧FIX⎫
     ⎩VAR⎭
```

Specifies the record format of the file. If you want to write variable-length records, enter VAR. If you want to write fixed-length records, omit the keyword or enter FIX.

```
RCSZ=n
```

Specifies the record size used when writing a unit record file to the printer, card punch, or a data set label diskette. The default value for this parameter depends on the type of device used. The default value for a printer is 132 bytes, the default for card punch is 80 bytes, and the default for data set label diskettes is 128 bytes.

A typical example of writing a unit record file is:

```
@W FILE='NEW ACCOUNTS',DEV=PR
```

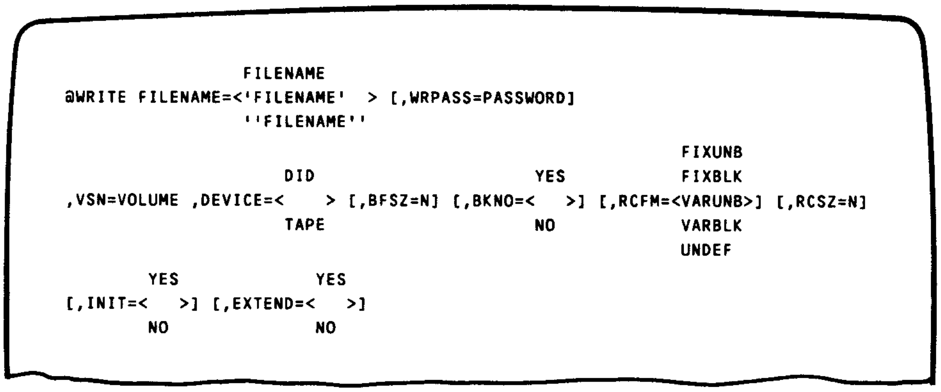
This command writes the current work-space file to the printer.

Format 4

To write the current work-space file to a tape, use this format:

```
@WRITE FILENAME=⎧filename    ⎫[,WRPASS=password]
                ⎨'filename'  ⎬
                ⎩''filename''⎭
     ,VSN=volume,DEVICE=⎧did ⎫[,BFSZ=n]⎡,BKNO=⎧YES⎫⎤
                        ⎩TAPE⎭         ⎣      ⎩NO ⎭⎦
    ⎡,RCFM=⎧FIXUNB⎫⎤[,RCSZ=n] ⎡,INIT=⎧YES⎫⎤⎡,EXTEND=⎧YES⎫⎤
    ⎢      ⎪FIXBLK⎪⎥          ⎣      ⎩NO ⎭⎦⎣        ⎩NO ⎭⎦
    ⎢      ⎨VARUNB⎬⎥
    ⎢      ⎪VARBLK⎪⎥
    ⎣      ⎩UNDEF ⎭⎦
```

The help screen format is:



```
                  FILENAME
@WRITE FILENAME=<'FILENAME'  > [,WRPASS=PASSWORD]
                  ''FILENAME''
                                                            FIXUNB
                     DID                     YES            FIXBLK
,VSN=VOLUME ,DEVICE=<    > [,BFSZ=N] [,BKNO=<   >] [,RCFM=<VARUNB>] [,RCSZ=N]
                     TAPE                    NO             VARBLK
                                                            UNDEF
       YES              YES
[,INIT=<   >] [,EXTEND=<   >]
       NO               NO
```

where:

```
FILENAME=(filename    )
         {'filename'  }
         (''filename'')
```

Specifies the name of the file to be written. The file name is the LBL name of the file and can be an existing LBL name, or, if you are creating the file now, you supply the LBL name with this parameter now. It can be from 1 to 44 alphanumeric characters. The file name must be enclosed in apostrophes or quotes when embedded spaces, commas, or parentheses are used in it.

```
WRPASS=password
```

Specifies a write password that restricts access to a file. The password is required only if a password entry for the file already exists in the system file catalog. The password can be from one to six alphanumeric characters.

```
VSN=volume
```

Specifies the volume serial number of the media where the specified file is to be written. The number can be from one to six alphanumeric characters. You must specify this parameter unless you previously entered the file in the system file catalog. If you don't know the vsn of the tape, you can specify the DEVICE parameter instead.

```
DEVICE={did }
       {TAPE}
```

Specifies the device you want to write to. did specifies the device address (did) of the tape you want to use. The first digit is the channel number; the second and third specify the hardware address. TAPE specifies that you want to write to a tape; however, you don't have to specify a specific tape.

```
BFSZ=n
```

Specifies the block size of the tape for I/O operations. The default is 256 bytes. Specifying a larger block size could reduce the number of disk I/O calls, causing your command to process faster. You would specify a block size different from the default when, for example, your command called for a block size other than 256.

```
BKNO={YES}
     {NO }
```

Specifies whether or not block numbers are created on the tape when the file is written. If you specify YES, block numbers are created on the tape. If you specify NO, they are not.

$$\underline{\mathrm{RCF}}\mathrm{M}=\left\{\begin{array}{l}\mathrm{FIXUNB}\\ \mathrm{FIXBLK}\\ \mathrm{VARUNB}\\ \mathrm{VARBLK}\\ \mathrm{UNDEF}\end{array}\right\}$$

Specifies the record format of the file. For descriptions of these record formats, refer to the consolidated data management concepts and facilities.

$\underline{\mathrm{RCS}}\mathrm{Z=n}$

Specifies the record size used when writing a file to tape. The default is 256 bytes. Specify this parameter only if the RCFM parameter equals FIXUNB or FIXBLK.

$$\underline{\mathrm{INI}}\mathrm{T}=\left\{\begin{array}{l}\mathrm{YES}\\ \mathrm{NO}\end{array}\right\}$$

Specifies whether the contents of a file are to be overwritten with new data being added. If you specify NO (the default), the old information remains intact and the new data is added to the end of the file. If you specify YES, the original file contents are overwritten with the new data, but the original file characteristics are not preserved. (The only exception would be if the original file had been created by using the default values for all the file parameters.) INIT assumes the default values for all files parameters. To overwrite a file by using the old file characteristics, use EXTEND=NO, not INIT. Otherwise, use INIT and specify your new file parameters. INIT applies only to MIRAM data files and tape files.

$$\underline{\mathrm{EX}}\mathrm{TEND}=\left\{\begin{array}{l}\mathrm{YES}\\ \mathrm{NO}\end{array}\right\}$$

Specifies whether the contents of a file are to be either overwritten or merged with new data. If you take the default YES, the existing file remains intact and the new data is added to the end. If you specify NO, the data is overwritten, but the file characteristics are preserved. To overwrite a file by using file characteristics, use the INIT=YES parameter instead of EXTEND. (The default values for EXTEND and INIT specify the same condition. The two parameters differ in how a file is overwritten with new data.) The EXTEND parameter is ignored if you specify INIT=YES. EXTEND applies only to MIRAM data files and tape files.

A typical example of writing a file to tape is:

```
@W FIL=''SALE.ITEMS'',VSN=SØØ071,DEV=TA
```

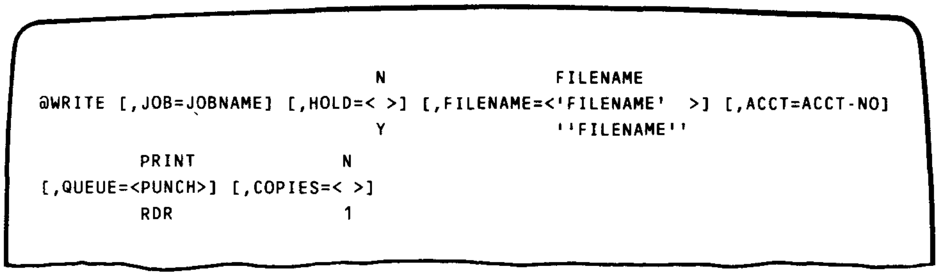
This command writes the current work-space file to the file SALE.ITEMS on tape volume S00071.

Format 5

To write the current work-space file to the spool file, use this format:

```
@WRITE [JOB=jobname] ⎡,HOLD=⎧NO ⎫⎤
                     ⎣      ⎩YES⎭⎦
   ⎡,FILENAME=⎧filename    ⎫⎤ [,ACCT=acct-no]
   ⎢          ⎨'filename'  ⎬⎥
   ⎣          ⎩''filename''⎭⎦
   ,QUEUE=⎧PRINT⎫ ⎡,COPIES=⎧n⎫⎤
          ⎨PUNCH⎬ ⎣        ⎩1⎭⎦
          ⎩RDR  ⎭
```

The help screen format is:



```
                           N               FILENAME
@WRITE [,JOB=JOBNAME] [,HOLD=< >] [,FILENAME=<'FILENAME'  >] [,ACCT=ACCT-NO]
                           Y               ''FILENAME''

       PRINT          N
[,QUEUE=<PUNCH>] [,COPIES=< >]
       RDR            1
```

where:

JOB=jobname

Specifies the name of the job that produced the spool file.

```
HOLD=⎧NO ⎫
     ⎩YES⎭
```

Specifies whether or not EDT writes the spool file in a HOLD state. If you want the file created in a HOLD state, enter HOLD=Y; if you do not want the file created in a HOLD state, enter HOLD=N. If you specify HOLD=Y, the file will not be printed or punched. If you specify QUEUE=RDR, EDT forces the HOLD option to NO.

```
FILENAME=⎧filename    ⎫
         ⎨'filename'  ⎬
         ⎩''filename''⎭
```

Specifies the logical file name of the spool file and must be the LBL name from your job control or the job name concatenated with the LFD name. It can be from 1 to 17 alphanumeric characters. The file name must be enclosed in apostrophes or quotes whenever embedded spaces, commas, or parentheses are used in it.

```
ACCT=acct-no
```

Specifies the account number of the job that produced the spool file you want to write. The account number may be from one to four alphanumeric characters. It's always optional.

```
QUEUE=⎧PRINT⎫
      ⎨PUNCH⎬
      ⎩RDR  ⎭
```

Indicates the spool queue you want to write your file to. You may enter PRINT, PUNCH, or RDR. PRINT indicates that the file is written to the printer queue. PUNCH indicates the file is written to the card punch queue. RDR indicates that the file is written to the card reader queue. The card reader queue in this case acts as a virtual card reader.

```
COPIES=⎧n⎫
       ⎩1⎭
```

Specifies the number of copies to be made of the spool file you want to process (where n = 1 to 255). The default is 1. This parameter is valid only when QUEUE=PRINT or QUEUE=PUNCH.

A typical example of writing a file to the spool file is:

```
@W FIL=PAYFILE,Q=PR,COP=2
```

This command writes the current work-space file to the file PAYFILE residing on the printer queue of the spool file. This example specifies that EDT print two copies of the file when the output writer spools it out.

Format 6

To write to the same module or file last accessed through a previous @READ or @WRITE command, use this format:

```
@WRITE
```

NOTE:

There is no help screen for format 6 of the WRITE command.

For example, let's say you update a module after reading it into EDT's work space with the command, @READ MO=MYMOD,FIL=MYFIL,VSN=REL080. After you make the necessary changes to the file, instead of including the three parameters on your @WRITE command, you simply key in @WRITE and EDT writes your module back to the module and file it came from.

Format 7

To write to the same module or file last accessed through a previous @READ or @WRITE command but written to now with any valid EDT command specified, use this format:

```
@WRITEΔ;Δvalid EDT command
```

NOTE:

There is no help screen for format 7 of the WRITE command.

where:

`;`
Specifies the same file parameters, with the exception of KEY, KKEY, and SHOW, used in the most recent @READ or @WRITE command.

`valid EDT command`
Specifies any valid EDT command that you want to combine with the WRITE command. Table 3–2 lists the valid commands that you can combine with the WRITE command.

For example:

```
@WRITE ; CHANGE 'ABC' TO 'DEF'
```

Writes the file in EDT's work space to the same module or file specified in the previous READ or WRITE command and changes all occurrences of ABC in the file to DEF.

Some typical examples of using WRITE combined with other commands are:

```
@W FIL=''REPORT 9'',VSN=D00031  C ALL '1980' TO '1981''
```

Writes the current work-space file to the file REPORT 9 on disk volume D00031 after changing every occurrence of 1980 to 1981. Note that ALL is not a parameter associated with the WRITE command, but is an EDT modifier, which we explain in Section 4.

```
@W MO=MYPROG,TY=P,FIL=MYFILE,VSN=D00034  INSERT '//'
```

Inserts two slashes (in columns 1 and 2) on every line in the current work-space file and then writes the current work-space file as the procedure MYPROG to the file MYFILE on disk volume D00034

3.19. COMBINING COMMANDS

As mentioned earlier, more than one command can be used in a single command line. Table 3–2 shows you which command combinations are valid, which are not allowed, and which are ignored by EDT because they are redundant.

Table 3–2. Command Combinations

	CHANGE	COPY	DELETE	FIND	FSTATUS	INSERT	LIST	MOVE	NUMBER	PRINT	PUNCH	READ	REMOVE	SEQUENCE	UPDATE	WRITE
CHANGE	R	▨	▨	▨	▨	R	▨	▨	R	▨	▨	▨	R	▨	▨	▨
COPY	▨	R	▨	▨	▨	▨	▨	R	N	▨	▨	▨	▨	R	▨	▨
DELETE	▨	▨	R	▨	N	▨	N	R	N	▨	N	N	▨	▨	▨	N
FIND	▨	▨	▨	R	▨	N	▨	▨	N	▨	▨	▨	▨	▨	▨	▨
FSTATUS	▨	▨	N	▨	N	▨	N	N	N	N	N	N	▨	▨	N	N
INSERT	R	▨	▨	N	▨	N	▨	▨	N	▨	▨	▨	N	N	▨	▨
LIST	▨	▨	N	▨	N	▨	R	N	N	N	N	N	▨	▨	N	N
MOVE	▨	R	R	▨	N	▨	N	R	N	▨	N	N	▨	▨	▨	N
NUMBER	R	N	N	N	N	N	N	N	N	▨	N	N	N	N	▨	N
PRINT	▨	▨	▨	▨	N	▨	N	▨	▨	R	N	N	▨	▨	▨	N
PUNCH	▨	▨	N	▨	N	▨	N	N	N	N	R	N	▨	▨	N	N
READ	▨	▨	N	▨	N	▨	N	N	N	N	N	N	▨	▨	N	N
REMOVE	R	▨	▨	▨	▨	N	▨	▨	N	▨	▨	▨	R	N	▨	▨
SEQUENCE	R	▨	▨	▨	▨	N	▨	▨	N	▨	▨	▨	N	N	▨	▨
UPDATE	▨	▨	▨	▨	N	▨	N	▨	▨	▨	N	N	▨	▨	R	N
WRITE	▨	▨	N	▨	N	▨	N	N	N	N	N	N	▨	▨	N	N

LEGEND:

R Redundant combinations

N Invalid combinations

▨ Valid combinations

4. Command Modifiers

4.1. INTRODUCTION

This section describes the EDT command modifiers. EDT command modifiers are 1-word modifiers that are used in combination with the EDT commands to enhance or restrict their performances. These modifiers allow you to execute EDT commands against strings, lines, or columns other than those implicitly specified in the basic EDT commands, or those without any modifiers specified.

Command modifiers must always be used with the EDT commands. They can't be used alone or with the EDT procedure file commands or directives. Not all EDT command modifiers can be used with every EDT command. For your convenience, the valid command and command modifier combinations are listed in Table 4-1.

Table 4—1. Valid Command Modifiers

	ALL	BY	COLUMN	FIRST	NOT	ON	SHOW	TO
@	N	N	N	N	N	N	N	N
CHANGE	▨	N	▨	▨	▨	▨	N	R
COPY	U	N	▨	▨	▨	N	N	R
DELETE	U	N	▨	▨	▨	▨	N	N
FIND	▨	N	▨	▨	▨	▨	N	N
FSTATUS	N	N	N	N	N	N	N	N
INPUT	N	N	N	N	N	N	▨	N
INSERT	N	N	▨	N	N	▨	N	N
LIST	N	N	N	N	▨	▨	▨	N
MOVE	U	N	▨	▨	▨	▨	N	R
NUMBER	N	R	▨	N	N	▨	N	N
PRINT	U	N	▨	N	▨	▨	▨	N
PUNCH	U	N	▨	N	▨	N	▨	N
READ	N	N	N	N	N	N	▨	N
REMOVE	▨	N	▨	▨	N	▨	N	N
SEQUENCE	U	R	N	N	N	▨	N	N
UPDATE	▨	N	▨	▨	N	▨	N	N
WRITE	N	N	N	N	N	N	▨	N

LEGEND:

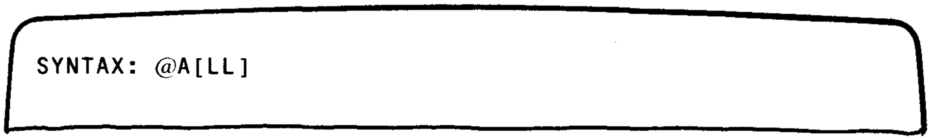
▨	Valid combinations
N	Invalid combinations
R	Required combinations
U	Unnecessary combinations, but valid

The following subsections contain detailed descriptions of the command modifiers available. These modifiers are presented in alphabetic sequence.

4.2. ALL

The ALL modifier repeats the action performed by a command throughout the implied or specified column range of each line in the work-space file that is processed. ALL is generally used in commands that act upon specified strings in the work-space file.

The help screen format is:



```
SYNTAX: @A[LL]
```

A typical example of using ALL is:

@CHANGE A 'OLD' TO 'NEW'
Changes every occurrence of OLD to NEW on each line in the current work-space file. Without ALL, the command changes only the first occurrence on each line.

B

4.3. BY

The BY command modifier specifies the increment to be used for sequence numbers inserted into input lines or existing lines in your current work-space file. BY is used only with the NUMBER and SEQUENCE commands and is required when using a sequence string.

The help screen format is:

```
SYNTAX: B[Y]
```

Some typical examples of using BY are:

@SEQ 'SU10' BY 2

Overlays columns 1 through 4 on the first line of the current work-space file with the sequence string SU10 and overlays columns 1 through 4 on subsequent (existing) lines in increments of 2 of that string

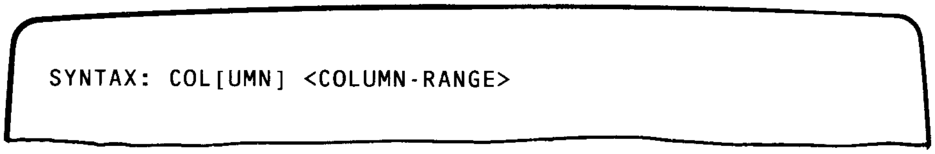
@NU 'X125' BY 25

Inserts sequence numbers into input lines, starting with the sequence string X125 and incrementing it by 25

4.4. COLUMN

The COLUMN modifier specifies the column range on which the command it is combined with is to act upon for each line of a file. COLUMN is used to limit the search operation and guarantees that the search string begins somewhere within the column limits specified.

The help screen format is:



```
SYNTAX: COL[UMN] <COLUMN-RANGE>
```

Some typical examples of the COLUMN modifier are:

@COL 5 INSERT '='
Inserts an equals sign in column 5 of every line in the current work-space file. Without COLUMN, this command inserts an equals sign in column 1 of every line.

@COL 70:80 CHANGE ALL 'MONTH' TO 'YEAR'
Changes every occurrence of MONTH to YEAR in columns 70 through 80 only on every line in the current work-space file. Without COLUMN, this command changes every occurrence of MONTH to YEAR on every line no matter where it appears.

@COL 10,12,14 DELETE '%'
Deletes every line in the current work-space file that contains a percent sign in columns 10, 12, or 14. Without COLUMN, the command deletes every line in the current work-space file that contains a percent sign in any column.

NOTE:

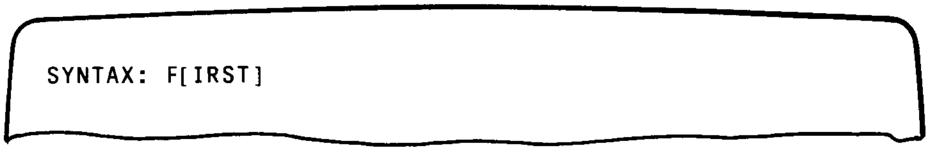
When more than one column range is specified, columns must be listed in ascending order (7, 14, 28, 35).

F

4.5. FIRST

The FIRST modifier directs EDT to stop processing the command it is combined with after the first line specified (implicitly or explicitly) in the command is completely processed.

The help screen format is:



```
SYNTAX: F[IRST]
```

Some typical examples of using FIRST are:

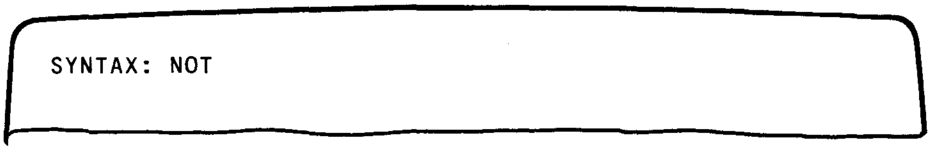
@PRINT F 'STREET'
Displays only the first line containing STREET. Without FIRST, it displays every line containing STREET.

@CHANGE F 'EDT' TO 'EDITOR'
Changes the first occurrence of EDT to EDITOR on the first line it appears on in the current work-space file. Without FIRST, the first occurrence of EDT on every line in the current work-space file is changed to EDITOR.

4.6. NOT

The NOT modifier executes the command it's combined with on all lines in the current work-space file that don't contain a specified string.

The help screen format is:



```
SYNTAX: NOT
```

Some typical examples of NOT are:

@PRINT NOT 'MAIN'
Displays all lines in the current work-space file that do not contain MAIN. Without NOT, the command displays all the lines that contain MAIN.

@COPY NOT 'GROUP' TO 50
Copies all lines (in the current work-space file) that do not contain GROUP to lines 50, 51, 52, etc. Without NOT, this command copies all the lines containing GROUP to lines 50, 51, 52, etc.

@DELETE NOT 'APPLE'
Deletes all lines that do not contain APPLE in the current work-space file. Without NOT, the command deletes all lines containing APPLE.

NOTE:

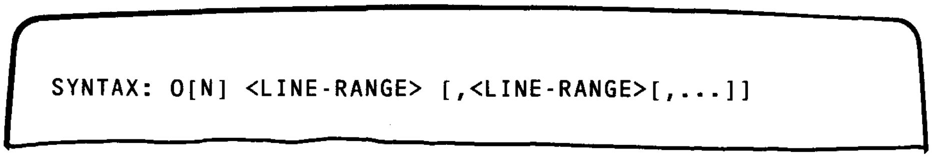
The NOT modifier can't be used when a change string is specified.

O

4.7. ON

The ON modifier specifies the line range on which the command it is combined with is to act upon.

The help screen format is:



```
SYNTAX: O[N] <LINE-RANGE> [,<LINE-RANGE>[,...]]
```

Some typical examples of ON are:

@O 4,7 INSERT '//'
Inserts two slashes beginning in column 1 on lines 4 and 7 of the current work-space file. Without ON, the command inserts two slashes beginning in column 1 on every line in the current work-space file.

@O 20:30 CHANGE ALL 'DUE' TO 'PAID'
Changes every occurrence of DUE to PAID on lines 20 through 30 only of the current work-space file. Without ON, the command changes every occurrence of DUE to PAID in the current work-space file.

@O 5:8 SEQ '10' BY 2
Overlays columns 1 and 2 on lines 5 through 8 of the current work-space file with sequence strings. The sequence string 10 overlays columns 1 and 2 of line 5, and increments of 2 of that string (12,14,16, etc) overlay columns 1 and 2 of subsequent lines in the specified line range (5:8).

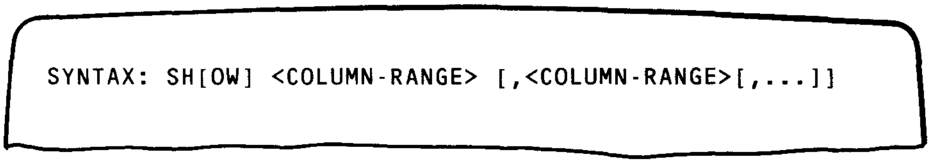
NOTE:

When more than one line range is specified, lines must be listed in ascending order.

4.8. SHOW

The SHOW modifier specifies the column range for the READ, WRITE, INPUT, LIST, PRINT, and PUNCH commands. SHOW used with LIST, PRINT, or PUNCH specifies which columns of the current work-space file are to be printed, displayed, or punched. SHOW used with the READ and INPUT commands causes partial records to be placed in the work-space file, and SHOW used with the WRITE command causes partial records to be written to a program library or data file. The SHOW command causes left-justification of data to column 1 and blank padding to the right to fill the record.

The help screen format is:



```
SYNTAX: SH[OW] <COLUMN-RANGE> [,<COLUMN-RANGE>[,...]]
```

Some examples of SHOW are:

@PRINT 'RON' COL 10:12 SH 65:70
Displays columns 65 through 70 of all lines in the current work-space file that contain RON in columns 10 through 12. Without SHOW, all lines in the current work-space file that contain RON in columns 10 through 12 are displayed in their entirety.

@READ FIL=MYFILE, VSN=D00029 SH 9:80
Reads the records in the file MYFILE, which resides on disk volume D00029, deletes columns 1 through 8 of each record, and then places the partial records in the work-space file using the current line number and increment. Because columns 1 through 8 are deleted from the records, each record is shifted eight columns to the left when it's entered in the work-space file, leaving columns 72 through 80 blank. The partial records now begin in column 1 instead of column 8. Without SHOW, whole records are read into the work-space file.

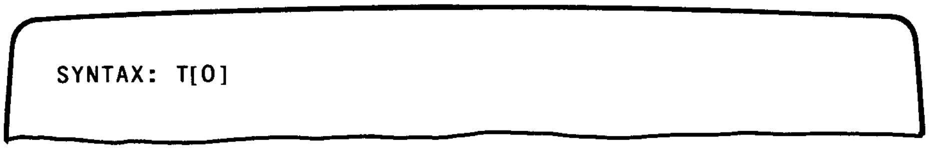
@WRITE FIL='FILE 6',VSN=D00029 SH 11:50
Writes columns 11 through 50 of each line in the work-space file to the file 'FILE 6' on disk volume D00029.

T

4.9. TO

The TO modifier specifies either the line destinations for moved or copied lines or a change string to replace a search string. TO is required in the MOVE, COPY, and CHANGE commands.

The help screen format is:



```
SYNTAX: T[O]
```

Some examples of TO are:

@MOVE 1 T 5
Moves line 1 to line 5

@COPY 2 T 75
Copies line 2 to line 75

@CHANGE 'APRIL' T 'MAY'
Changes the first occurrence of APRIL on every line in the current work-space file to MAY

5. EDT Procedure Files

5.1. INTRODUCTION

EDT enables you to divide the EDT work-space file into a maximum of 10 subfiles (procedure files) as shown in Figure 5-1. Notice that there is a main work file (the work-space file you are already familiar with) and nine other procedure files.

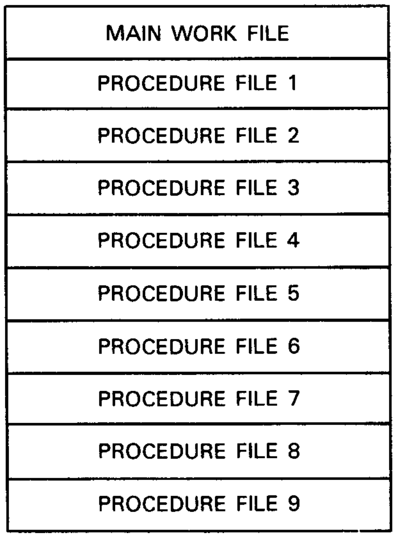


Figure 5-1. EDT Work-Space File

EDT procedure files are basically like the main work file; they consist of data and/or EDT commands. However, unlike the main work file, EDT procedure files can be executed. The commands or data entered in them can be executed against the main work file or any of the other defined procedure files.

Initially, all EDT procedure files are empty and you may create or update them any time during an EDT session. Every time a new EDT procedure file is created, EDT assigns the line number 1 to the first line of the procedure file and increments subsequent line numbers by 1. As many command lines or data lines as desired can be read or keyed in to define an EDT procedure file. However, if you want to enter a command line as text (a command that is to be executed when the procedure file is executed), it must be preceded by a double at sign (@@), which indicates that the entry is text. This prevents lines that contain commands from being executed as they are keyed in.

EDT procedure files don't have to be executed. They can be used independently of each other to create, edit, and update multiple files concurrently during an EDT session. Up to 10 files can be located in the work space at one time.

The following subsections explain how to use EDT procedure files and explain the commands used to create, manipulate, or execute EDT procedure files. For your convenience, the EDT procedure file commands are summarized in Table 5-1.

Table 5—1. Procedure File Commands

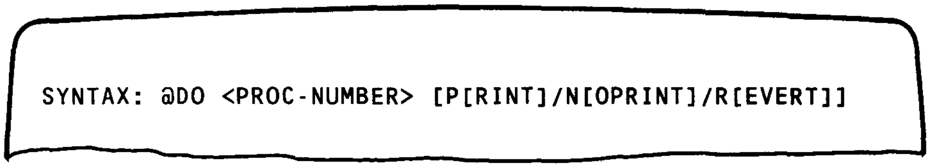
Command	Function
DO	Executes a procedure file
END	Terminates procedure file definition
GOTO	Permits branching within a procedure file
INPUT	Loads and executes a procedure file
NOP	Permits extra line numbers and comments to be inserted into a procedure file
PROC	Begins procedure file definition
RETURN	Terminates procedure file execution

5.2. EXECUTING A PROCEDURE FILE (@DO)

The DO command executes an EDT procedure file. The format is:

```
@DO proc-number ⎡⎧PRINT  ⎫⎤
                ⎢⎨NOPRINT⎬⎥
                ⎣⎩REVERT ⎭⎦
```

The help screen format is:



```
SYNTAX: @DO <PROC-NUMBER> [P[RINT]/N[OPRINT]/R[EVERT]]
```

where:

proc-number

Specifies which EDT procedure file is to be executed. It is a single digit integer from 1 to 9.

PRINT

Specifies each line of the EDT procedure file is to be displayed on the workstation screen before it is executed.

NOPRINT

Specifies each line of the EDT procedure file is not to be displayed before it is executed.

REVERT

Sets the print status to the state it was in (either PRINT or NOPRINT) when the current procedure file was entered.

If PRINT, NOPRINT, or REVERT is not specified, the display of lines depends on whether or not the @DO is being issued from another EDT procedure file. If the @DO is issued from another EDT procedure file, the print option currently active in the calling EDT procedure file is used. If the @DO is issued from the main work file, NOPRINT is assumed.

NOTE:

You also can use the DO command to change the print status without executing a procedure file. Enter @DO and the appropriate print status and do not specify a proc-number.

Here are some typical examples of the DO command:

```
@DO 3 P
```

Executes procedure file 3 and displays each line of that procedure file before it is processed

```
@DO 7 N
```

Executes procedure file 7 but does not display its lines before they are processed

```
@DO P
```

Turns on the line display process of the current procedure file

NOTES:

1. *During the execution of an EDT procedure file, input is taken from that procedure file. This input is handled in the same way as if it was keyed in at the workstation.*
2. *Once the execution of an EDT procedure file is completed, control is returned to the file that specified it. This file may be the main work file or any other defined EDT procedure file. The line following @DO in the file then becomes the current line.*
3. *If you execute a procedure file that contains an @BLOCK screen command, you will get errors.*

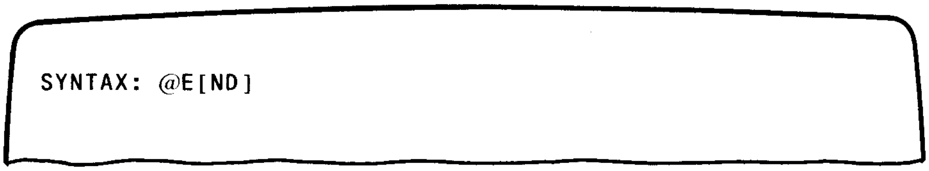
5.3. TERMINATING PROCEDURE FILE DEFINITION (@END)

The END command ends procedure file definition.

The format is:

```
@ END
```

The help screen format is:



```
SYNTAX: @E[ND]
```

@END marks an EDT procedure file as completely defined and returns control to the file containing the @PROC that specified it. This file may be the main work file or any of the defined EDT procedure files. The current line number is then set to the line number last displayed at the time the corresponding PROC command was issued (the line number of the line containing @PROC).

NOTE:

@END cannot be keyed in while in the main work file.

G

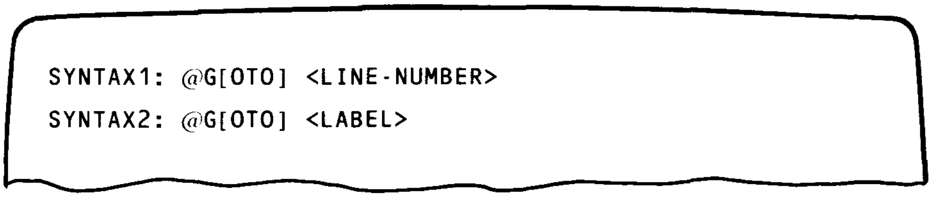
5.4. BRANCHING WITHIN A PROCEDURE FILE (@GOTO)

The GOTO command causes a branch to another command line in an EDT procedure file.

The format is:

```
@GOTO{label}
     {line }
```

The help screen format is:



```
SYNTAX1: @G[OTO] <LINE-NUMBER>
SYNTAX2: @G[OTO] <LABEL>
```

where:

label
: Specifies by label the next line to be processed.

line
: Specifies by line number the next line to be processed.

The following are examples of the GOTO command.

If you want to branch to line 7

```
7.0000▷@(XYZ) PRINT 10:20
```

you can do so by keying in either of the following commands:

```
3.0000▷@G XYZ
```

or

```
3.0000▷@G 7
```

Or, for instance, if you want to branch to line 35

```
35.ØØØØ▷@(SRW181) MOVE 4Ø TO 45
```

you can use either of the following commands:

```
15.ØØØØ▷@G SRW181
```

or

```
15.ØØØØ▷@G 35
```

INP

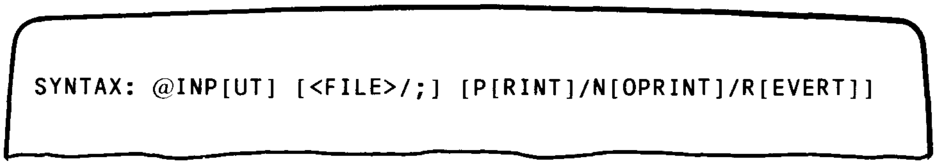
5.5. LOADING AND EXECUTING PROCEDURE FILES (@INPUT)

The INPUT command permits user-written EDT procedure files to be loaded and executed in a single command. These EDT procedure files may be SAT or MIRAM library modules, or MIRAM program libraries or data files.

The format is:

```
@INPUT file-parameters ⎡⎧PRINT  ⎫⎤
                       ⎢⎨NOPRINT⎬⎥
                       ⎣⎩REVERT ⎭⎦
```

The help screen format is:



```
SYNTAX: @INP[UT] [<FILE>/;] [P[RINT]/N[OPRINT]/R[EVERT]]
```

where:

file-parameters
: Specifies those parameters needed to read a library module, program library, or data file into the EDT work-space file. These parameters are discussed in 3.14 (the READ command).

PRINT
: Specifies that each line of the EDT procedure file is to be displayed on the workstation screen before it is executed.

NOPRINT
: Specifies that each line of the EDT procedure file is not to be displayed before it is executed.

REVERT
: Sets the print status to the state it was in (either PRINT or NOPRINT) when the current procedure file was entered.

If PRINT, NOPRINT, or REVERT is not specified, the display of lines depends on whether or not the INPUT command is being issued from another EDT procedure file. If the INPUT command is issued from another EDT procedure file, the print option currently active in the calling EDT procedure file is used. If the INPUT command is issued from the main work file, NOPRINT is assumed.

When an INPUT command is issued, EDT creates a procedure file 10 in the EDT work-space file, loads the file to be executed into procedure file 10, and then executes it. The file remains in procedure file 10 until another INPUT command is issued, replacing it with a new file.

NOTE:

You can access procedure file 10 using the INPUT command only, not by using the normal procedure file commands.

A typical example of the INPUT command is:

```
@INP FIL='ACCT FILE',VSN=D00029 P
```

Loads and executes the file ACCT FILE on volume D00029 as an EDT procedure file (proc 10). Each line of the procedure file is displayed before it is executed because the PRINT option was specified.

NOP

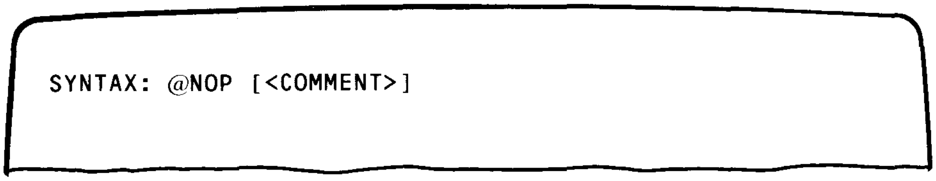
5.6. ENTERING EXTRA LINE NUMBERS AND COMMENTS (@NOP)

The NOP (no-operation) command is used to enter extra lines for branching or to enter comments into EDT procedure files.

The format is:

```
@NOP[ comment]
```

The help screen format is:



```
SYNTAX: @NOP [<COMMENT>]
```

where:

`comment`
: Specifies a comment line to be inserted on a line in an EDT procedure file. If the comment is omitted, an extra line number is inserted into the file for branching.

A typical example of the NOP command is:

5.0000@NOP THIS IS A COMMENT.
Places the comment THIS IS A COMMENT on line 5 of the file. (The comment is not recognized as data.)

NOTE:

The NOP command may also be used in the main work file.

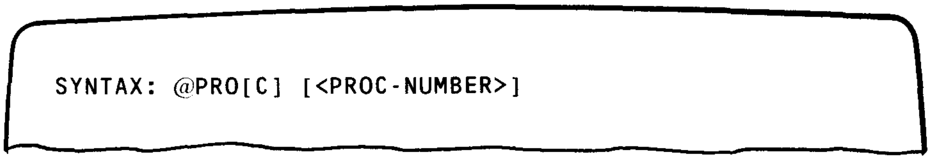
5.7. DEFINING A PROCEDURE FILE (@PROC)

The PROC command displays the number of the procedure file you are currently using or makes a procedure file available for you to use.

The format is:

```
@ PROC [proc-number]
```

The help screen format is:



```
SYNTAX: @PRO[C] [<PROC-NUMBER>]
```

where:

`proc-number`
: Is an integer from 1 to 9 that specifies the number of the EDT procedure file that is to be made available for use (as the current work-space file). When proc-number is omitted, the number of the current procedure file is displayed.

Some typical examples of the PROC command are:

@PRO
Displays the number of the procedure file currently in use

@PRO 9
Makes procedure file 9 the current work-space file in the EDT session

If the procedure file you specify is empty, line 1.0000 is displayed on the workstation screen. You then key in data or a command to begin defining the file. However, if the procedure file was previously defined (it already contains lines), the first line number displayed is then the line number that was current the last time the file was accessed.

NOTE:

Commands keyed in after the PROC command is executed affect only the contents of the current EDT procedure file. They do not affect the contents of the main work file or other defined EDT procedure files.

RET

5.8. TERMINATING PROCEDURE FILE EXECUTION (@RETURN)

The RETURN command terminates the execution of the EDT procedure file in which it appears.

The format is:

```
@RETURN
```

The help screen format is:

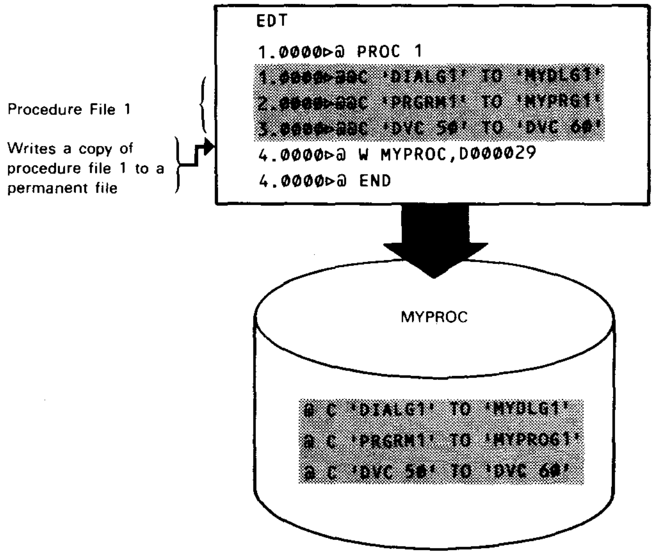
```
SYNTAX: @RET[URN]
```

When the execution of an EDT procedure file is terminated via the RETURN command, control is returned to the line following the DO command that specified the procedure file to be executed.

5.9. USING EDT PROCEDURE FILES

The following sample EDT session shows you how to create, save, access, and execute an EDT procedure file. We will use this procedure file to update a job control stream in the main work-space file.

EDT SESSION 1 . . . . Creating and Saving an EDT Procedure File

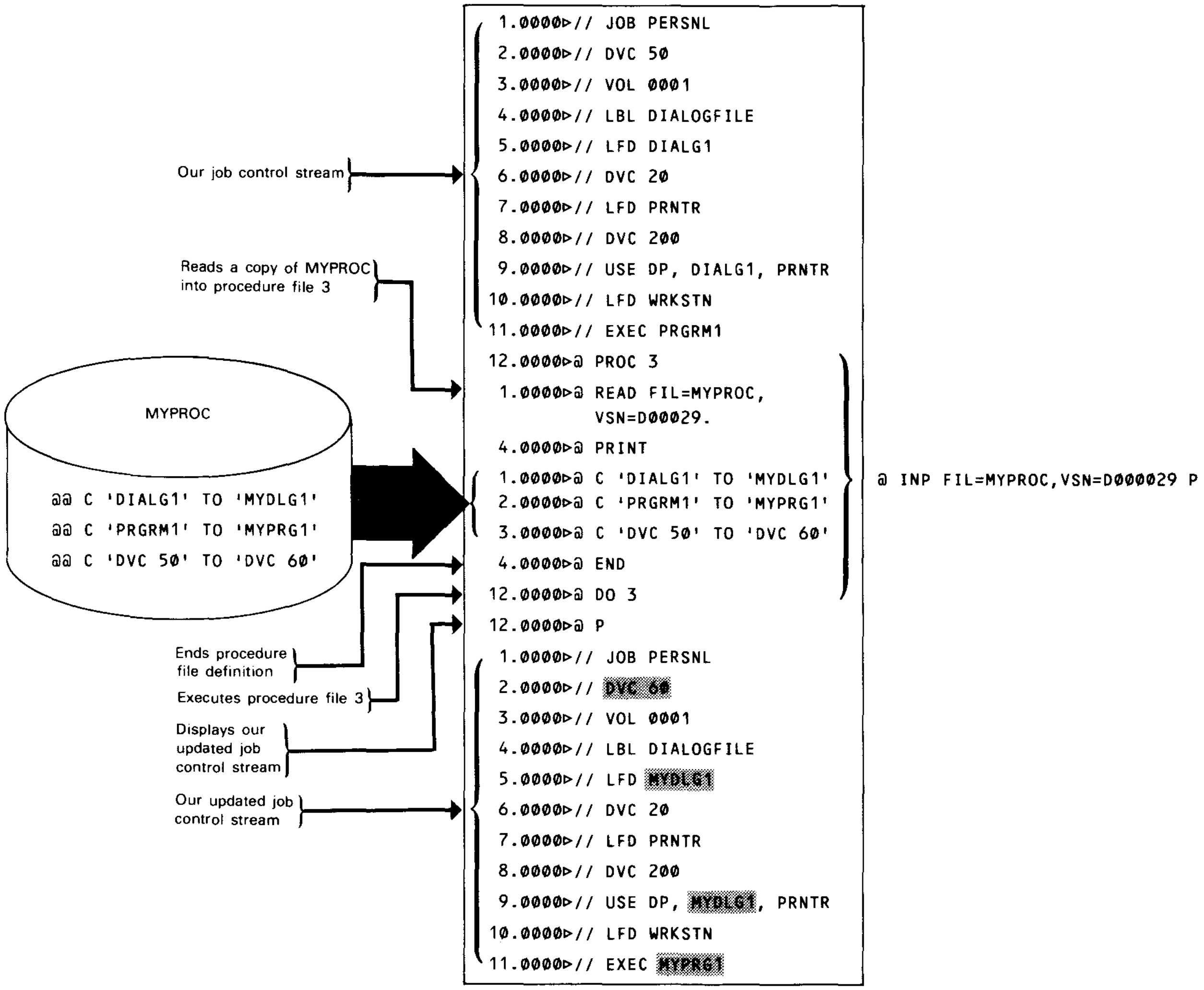


```
EDT
1.0000▷@ PROC 1
1.0000▷@@C 'DIALG1' TO 'MYDLG1'
2.0000▷@@C 'PRGRM1' TO 'MYPRG1'
3.0000▷@@C 'DVC 50' TO 'DVC 60'
4.0000▷@ W MYPROC,D000029
4.0000▷@ END
```

```
@ C 'DIALG1' TO 'MYDLG1'
@ C 'PRGRM1' TO 'MYPROG1'
@ C 'DVC 50' TO 'DVC 60'
```

In this session we created an EDT procedure file (PROC 1). Then we saved the contents of PROC 1 (via the WRITE command) in the permanent file MYPROC on disk volume D00029. The shaded portion denotes our saved procedure file. Notice that only data lines (command lines preceded by a double at sign (@@)) are written to the permanent file. Note, too, that EDT deletes one of the two at signs before saving each line. This enables the command lines to be executed easily via the DO or INPUT commands when the procedure file is read back into the work-space file.

EDT SESSION 2 . . . . Loading and Executing an EDT Procedure File


Our job control stream
Reads a copy of MYPROC into procedure file 3
MYPROC
@@ C 'DIALG1' TO 'MYDLG1'
@@ C 'PRGRM1' TO 'MYPRG1'
@@ C 'DVC 5Ø' TO 'DVC 6Ø'
Ends procedure file definition
Executes procedure file 3
Displays our updated job control stream
Our updated job control stream

```
 1.ØØØØ▷// JOB PERSNL
 2.ØØØØ▷// DVC 5Ø
 3.ØØØØ▷// VOL ØØØ1
 4.ØØØØ▷// LBL DIALOGFILE
 5.ØØØØ▷// LFD DIALG1
 6.ØØØØ▷// DVC 2Ø
 7.ØØØØ▷// LFD PRNTR
 8.ØØØØ▷// DVC 2ØØ
 9.ØØØØ▷// USE DP, DIALG1, PRNTR
1Ø.ØØØØ▷// LFD WRKSTN
11.ØØØØ▷// EXEC PRGRM1
12.ØØØØ▷@ PROC 3
 1.ØØØØ▷@ READ FIL=MYPROC,
          VSN=DØØØ29.
 4.ØØØØ▷@ PRINT
 1.ØØØØ▷@ C 'DIALG1' TO 'MYDLG1'      @ INP FIL=MYPROC,VSN=DØØØØ29 P
 2.ØØØØ▷@ C 'PRGRM1' TO 'MYPRG1'
 3.ØØØØ▷@ C 'DVC 5Ø' TO 'DVC 6Ø'
 4.ØØØØ▷@ END
12.ØØØØ▷@ DO 3
12.ØØØØ▷@ P
 1.ØØØØ▷// JOB PERSNL
 2.ØØØØ▷// DVC 6Ø
 3.ØØØØ▷// VOL ØØØ1
 4.ØØØØ▷// LBL DIALOGFILE
 5.ØØØØ▷// LFD MYDLG1
 6.ØØØØ▷// DVC 2Ø
 7.ØØØØ▷// LFD PRNTR
 8.ØØØØ▷// DVC 2ØØ
 9.ØØØØ▷// USE DP, MYDLG1, PRNTR
1Ø.ØØØØ▷// LFD WRKSTN
11.ØØØØ▷// EXEC MYPRG1
```

*You can also use the INPUT command to load and execute an EDT procedure instead of using READ and DO commands.

In this session we updated a job control stream in our main work file by executing an EDT procedure file against it. We loaded and executed a copy of the procedure file we created in session 1. Here, the shaded portions denote the changes made to our original job control stream after the procedure file was executed. If you had specified the single INPUT command, @INPFIL=MYPROC, VSN=D00029 P, EDT would've loaded and executed MYPROC without your having specified the PROC, READ, END, and DO commands.

6. EDT Variables

6.1. INTRODUCTION

EDT provides 10 EDT variables for use with the EDT procedure files. These variables are used to hold such values as line numbers, column numbers, and strings (of up to 50 characters). They are written in the format #Gn (where n is an integer from 0 to 9). These variables give flexibility in structuring commands.

Initially, all 10 EDT variables consist of null strings. However, values may be assigned to the EDT variables at any time before or during the execution of an EDT procedure file.

Before executing a command, EDT scans the command line for an EDT variable (e.g., #Gn). If a valid EDT variable is found, the variable is replaced with its string value and then the command is executed. Because the value of an EDT variable is placed in the command before it is executed, it is possible to assign any part of a valid EDT command to a variable.

If the symbol #G is to appear as data in the command line, it must be written ##G so as not to be confused with a variable symbol.

AS

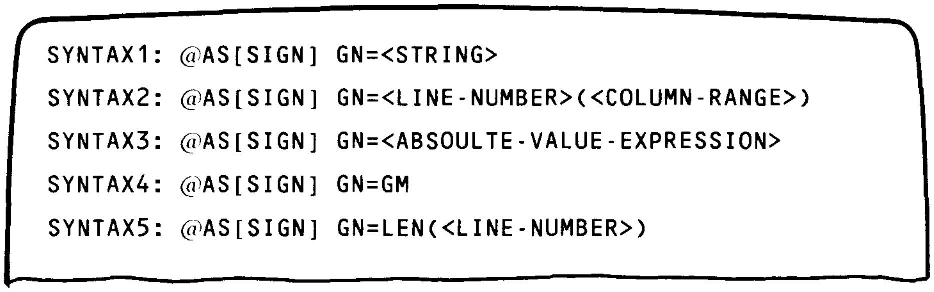
6.2. ASSIGNING VALUES TO EDT VARIABLES (@ASSIGN)

The ASSIGN variable assigns values to EDT variables.

The format is:

```
@ASSIGN Gn= ⎧'string'[*n]⎫
            ⎪n(x:y)      ⎪
            ⎨n[±m]       ⎬
            ⎪Gm          ⎪
            ⎩LEN(n)      ⎭
```

The help screen format is:



```
SYNTAX1: @AS[SIGN] GN=<STRING>
SYNTAX2: @AS[SIGN] GN=<LINE-NUMBER>(<COLUMN-RANGE>)
SYNTAX3: @AS[SIGN] GN=<ABSOULTE-VALUE-EXPRESSION>
SYNTAX4: @AS[SIGN] GN=GM
SYNTAX5: @AS[SIGN] GN=LEN(<LINE-NUMBER>)
```

where:

`Gn`
Specifies the EDT variable, where n must be an integer from 0 to 9.

`'string'`
Specifies a string to be assigned to an EDT variable. The string must be enclosed in apostrophes. Multiple copies of the string may be specified by using the string multiplication factor *n, where n is the number of times the string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

`n(x:y)`
Specifies the contents in columns x through y of line n to be assigned to an EDT variable.

`n[±m]`
Specifies a number to be assigned to an EDT variable. The number n alone or the number n plus or minus the number m is assigned to an EDT variable. (Only the absolute values for the numbers are assigned.)

`Gm`

Specifies that the value assigned to the EDT variable Gm is also assigned to another EDT variable (Gn). m must be an integer from 0 to 9.

`LEN(n)`

Specifies the length of a line to be assigned to an EDT variable. n is the work-space line number.

Some typical examples of the ASSIGN variable are:

@A G1='ABCD'

Assigns ABCD to the EDT variable G1

@A G5='#G1(2:3)'

Assigns the contents of columns 2 through 3 of variable G1 to variable G5. If variable G1 contains BC in columns 2 through 3, as shown in the first example, then G5 is assigned the value BC.

@A G4=12.5 (10:16)

Assigns the contents in columns 10 through 16 of line 12.5 to the variable G4

@A G7=1-5

Assigns the number 4 (-5+1) to the variable G7. (Note that the absolute value of the number (| -4 |) is always used.)

@A G9=G1

Assigns the value that is assigned to the variable G1 to the variable G9.

@A G2=LEN(12)

Assigns the length of line 12 to the variable G2

DI

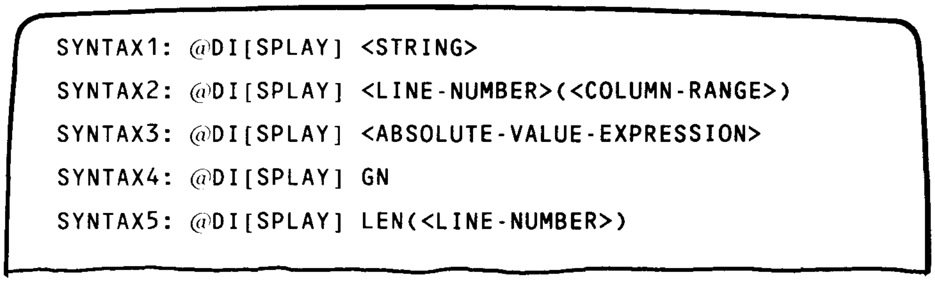
6.3. DISPLAYING VALUES OF EDT VARIABLES (@DISPLAY)

The DISPLAY variable command displays a specified string, number, or value of an EDT variable on the workstation screen.

The format is:

```
@DISPLAY  ⎧'string'[*n]⎫
          ⎪n(x:y)      ⎪
          ⎨n[±m]       ⎬
          ⎪Gm          ⎪
          ⎩LEN(n)      ⎭
```

The help screen format is:



```
SYNTAX1: @DI[SPLAY] <STRING>
SYNTAX2: @DI[SPLAY] <LINE-NUMBER>(<COLUMN-RANGE>)
SYNTAX3: @DI[SPLAY] <ABSOLUTE-VALUE-EXPRESSION>
SYNTAX4: @DI[SPLAY] GN
SYNTAX5: @DI[SPLAY] LEN(<LINE-NUMBER>)
```

where:

`'string'`
: Specifies a string to be displayed on the workstation screen. Multiple copies of the string may be specified by using the string multiplication factor *n, where n is the number of times the string is multiplied. After multiplication, the product of the number of characters in the string being multiplied and the string multiplication factor can't exceed 50.

`n(x:y)`
: Specifies the contents of columns x through y of line n to be displayed.

n[±m]
: Specifies a number to be displayed on the workstation screen. The number n alone or the number n plus or minus the number m is displayed. (Only the absolute values for the numbers are displayed).

`Gm`

Specifies that the value assigned to the EDT variable Gm is to be displayed on the workstation screen.

`LEN(n)`

Specifies the length of a line to be displayed. n is the work-space line number.

NOTE:

All displays will be 50 characters or less in length.

Some typical examples of the DISPLAY variable command are:

```
@DI 'ANN'
```

Displays the string ANN on the workstation screen

```
@DI 12.5(10:16)
```

Displays the contents in columns 10 through 16 of line 12.5 on the workstation screen

```
@DI 12+8
```

Displays 20 on the workstation screen

```
@DI LEN(25)
```

Displays the length of line 25 on the workstation screen

```
@DI G9
```

Displays the value assigned to the variable G9 on the workstation screen. For example, if the value assigned to the variable G9 is 'CRACKERS', when @DI #G9 is issued, CRACKERS is displayed on the screen.

The DISPLAY variable command is useful when displaying the contents of an EDT variable at a certain point in a debugging session or to display some messages during the execution of a procedure file. It is also used to display or format a line without actually creating a data line in the work-space file.

IF

6.4. BRANCHING ON CONDITION (@IF)

The IF variable permits an EDT command or an EDT procedure file command to be executed based on some condition. @IF can be used in both the main work file and EDT procedure files.

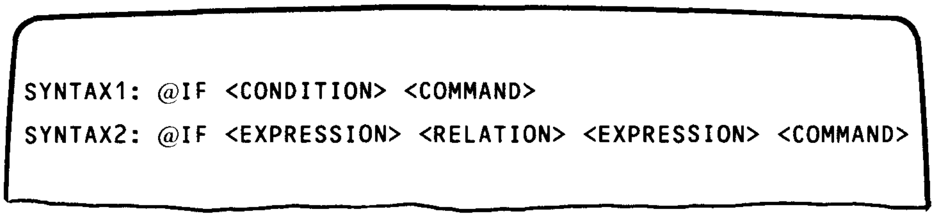
The formats are:

```
@IF.condition.command
```

or

```
@ IF expression relation expression command
```

The help screen format is:



```
SYNTAX1: @IF <CONDITION> <COMMAND>
SYNTAX2: @IF <EXPRESSION> <RELATION> <EXPRESSION> <COMMAND>
```

where:

`condition`
Defines the operands used to test the success of locating a search string in the last search command or an error in a previous command. The operands are:

.T. for a successful search
.F. for an unsuccessful search
.E. for the existence of an error in a previous command. A test for errors causes the error indicator to be reset. This indicator can also be reset by @SET CLEAR.

`expression`
Defines an EDT variable expression. (See 6.2.)

relation

Defines a relational operator. They are:

<	less than
>	greater than
=	equal to
< =	less than or equal to
> =	greater than or equal to
< >	not equal to

command

Is an EDT command or EDT procedure file command.

The best way to explain the use of the IF variable is to show it in some examples:

```
@CHANGE 'RIGHT' TO 'WRONG'
@IF .F. CHANGE 'WRONG' TO 'RIGHT'
```

The IF command tests whether the search string RIGHT was located. If it was not located, then the command @CHANGE 'WRONG' TO 'RIGHT' is executed.

```
@IF #G6 > = 25 PRINT 'ACCOUNT'
```

Directs EDT to display all the lines containing ACCOUNT if the value assigned to the variable G6 is greater than or equal to 25

```
@IF #G4 < > 9 ASSIGN G4=9
```

Directs EDT to assign 9 to the variable #G4 if it is not already assigned to it

6.5. USING EDT VARIABLES

Using EDT variables increases flexibility and efficiency of the commands associated with them.

The following examples will give you some ideas as to how you can use the EDT variables to your advantage. Assume these values for the variables:

```
@ ASSIGN G1=1
@ ASSIGN G2=5
@ ASSIGN G3='SEARCH'
@ ASSIGN G4=G1 + G2
@ ASSIGN G5='COL #G2 INSERT '*' '
```

@DELETE #G3
Deletes all lines that contain SEARCH (#G3)

@ON #G1:#G4 PRINT #G3
Displays all lines from line 1 (#G1) through line 6 (#G4) that contain SEARCH (#G3)

@ON #G2 COL #G1:#G4 CHANGE '#G3' TO 'LOCATE'
Changes SEARCH (#G3) to LOCATE in columns 1 (#G1) through 6 (#G4) on line 5 (#G2)

@ON 1:20 #G5
Executes the command COL #G2 INSERT '*' (#G5) against lines 1 through 20. Specifically, it inserts an asterisk in column 5 (#G2) on lines 1 through 20.

You can also use the EDT variables to specify a portion of a string.

For instance, if

```
#G7 = 'ALPHABET'
```

you could specify the ALPHA portion of the string by keying in the column range it occupies after the variable, like this:

```
#G7(1:5)
```

NOTE:

If the specified ending column is greater than the actual length of the variable, blanks are appended to the right side as needed:

```
#G7(6:12) = 'BET    '
```

7. Directives

7.1. INTRODUCTION

Another set of commands known as the EDT directive commands are also available to you. They serve as the interface between you, EDT, and the operating system. These are the commands that enable you to execute commands, access specialized language editors, and terminate EDT. They are also used to instruct EDT to adjust defaulted parameters. The directives CHECK, COBOL, DROP, EFP, FORMAT, HALT, RPG, SET, and SYSTEM are described in the following subsections.

CHE

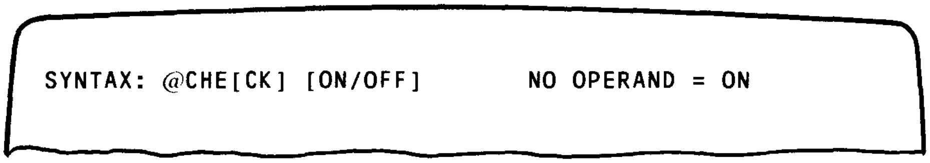
7.2. DISPLAYING PROCESSED LINES ON THE WORKSTATION (@CHECK)

The CHECK directive determines if lines processed as a result of an EDT command are to be displayed on the workstation screen.

The format is:

```
@CHECK ⎡⎧ON ⎫⎤
       ⎣⎩OFF⎭⎦
```

The help screen format is:



```
SYNTAX: @CHE[CK] [ON/OFF]        NO OPERAND = ON
```

where:

ON
: Specifies that the processed lines are to be displayed.

OFF
: Specifies that the processed lines are *not* to be displayed.

If you issue an @CHECK=ON directive, EDT displays the lines processed as a result of another EDT command according to your specification for the SCRDSPLY parameter on the @SET directive. If SCRDSPLY=TRUNCATE, EDT displays one record per one screen line and truncates lines if necessary; if SCRDSPLY=FOLD, EDT displays one record per two screen lines and doesn't truncate lines.

After EDT prints a full screen or the last screen, you have four options:

1. you may update the displayed lines and transmit them to the EDT work space;

2. you may continue the display of processed lines by pressing the F19 function key;

3. you may prevent the display of processed lines by pressing function key F1 or F18; or

4. you may terminate the @CHECK=ON directive by pressing the F2 function key.

For more information on the use of these and other function keys, see Section 9 and the interactive services commands and facilities user guide/programmer reference.

Here is a typical example of the CHECK directive:

If the work-space file contained the following data lines

```
1.0000 BEAR
2.0000 CAT
3.0000 ELEPHANT
4.0000 CAT
```

and the following commands were issued

```
5.0000▷@CHE ON
5.0000▷@CHANGE 'CAT' TO 'LION'
```

these lines would be displayed on the workstation

```
2.0000▷LION
4.0000▷LION
```

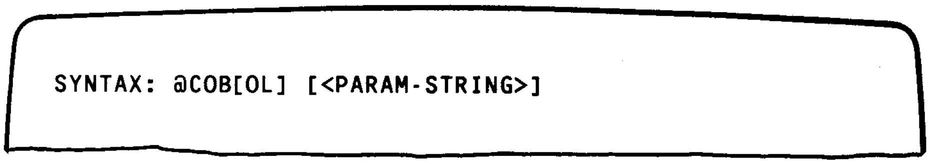
COB

7.3. ACTIVATING THE COBOL EDITOR (@COBOL)

The COBOL directive activates the COBOL editor (assuming your system is configured with a COBOL editor) to create and maintain ANSI '74 COBOL source modules. During an EDT session, you activate the COBOL editor by keying in the following directive:

```
@COBOL
```

The help screen format is:



```
SYNTAX: @COB[OL] [<PARAM-STRING>]
```

If you aren't currently in an EDT session, to activate the COBOL editor you must first key in EDT to activate EDT and then follow that command with the COBOL directive:

```
EDTΔ@COBOL
```

If you are already using the RPG II editor, end your RPG II session before activating the COBOL editor.

For further information about the COBOL directive, see the COBOL editor user guide/programmer reference.

7.4. DELETING THE CONTENTS OF THE EDT WORK-SPACE FILE (@DROP)

The DROP directive deletes all lines in the entire EDT work-space file (i.e., the main work file and all defined EDT procedure files).

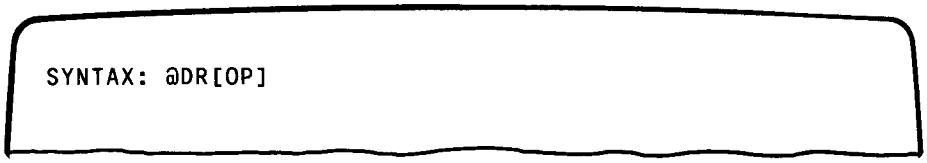
The format is:

```
@DROP
```

NOTE:

The difference between the DROP directive and the DELETE command is that @DROP deletes all lines in the entire work-space file, whereas @DELETE deletes only the lines in the current work-space file (i.e., the file you are currently using).

The help screen format is:



```
SYNTAX: @DR[OP]
```

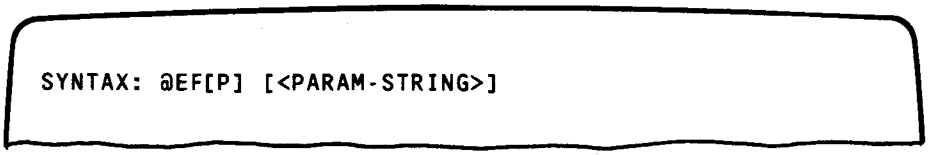
EFP

7.5. ACTIVATING THE ERROR FILE PROCESSOR (@EFP)

The EFP directive activates the error file processor so that you can correct your source code without waiting for your language processor to provide its output listing. During an EDT session, you activate the error file processor by keying in the following directive:

```
@EFP
```

The help screen format is:



```
SYNTAX: @EF[P] [<PARAM-STRING>]
```

If you aren't currently in an EDT session, to activate the error file processor you must first key in EDT to activate EDT and then follow that command with the EFP directive:

```
EDTΔ@EFP
```

For further information about error file processing, see Section 11 of this manual.

7.6. INTERFACING WITH RPGEDT OR COBEDT (@FORMAT)

Depending on the subeditor you're working with (RPGEDT or COBEDT), the @FORMAT directive has two functions. However, it is valid only when you're in a subeditor session.

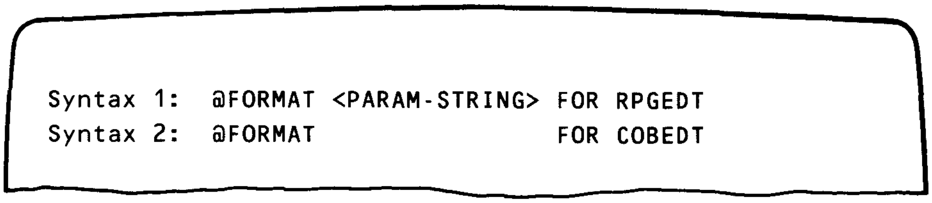
For RPGEDT, the @FORMAT directive lets you change the display format type and/or switch RPGEDT from update mode to create mode. For COBEDT, @FORMAT lets you return to COBEDT after you temporarily exited from COBEDT to EDT through the CMD continuation code during a COBEDT session. The formats for the @FORMAT directive are:

```
@FORMAT parameter string (for RPGEDT)
```

or

```
@FORMAT                  (for COBEDT)
```

The help screen format is:



```
Syntax 1:  @FORMAT <PARAM-STRING> FOR RPGEDT
Syntax 2:  @FORMAT                FOR COBEDT
```

For more information on the @FORMAT directive, see the appropriate subeditor manual.

7.7. TERMINATING EDT (@HALT)

The HALT directive terminates the EDT session. It releases the work-space file (i.e., the main work file and all defined EDT procedure files).

The format is:

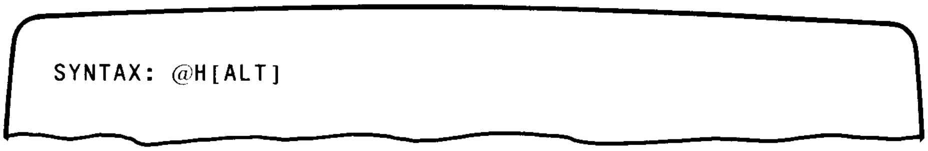
```
@HALT
```

NOTES:

1. *If you haven't written the contents of the work-space file to a permanent file when @HALT is entered at the workstation, the message ED003 EDITED FILES NOT SAVED, TERMINATE (Y,N)? is displayed to remind you to do so. If you answer Y, EDT is terminated without saving the contents of your work-space file; however, if you answer N, the current work-space line number is displayed so that you can enter a WRITE command to save it. Once the file is saved, you reissue the HALT directive to terminate EDT.*

2. *If you are using the error file processor and issue an @HALT directive, EDT first terminates the error file processor automatically, then terminates itself.*

The help screen format is:



```
SYNTAX: @H[ALT]
```

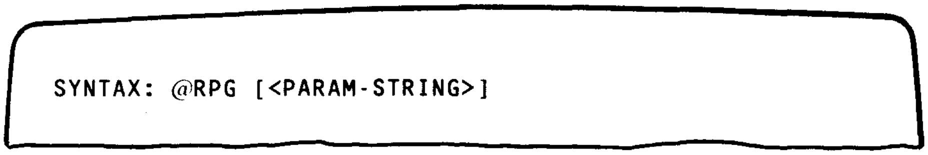
RPG

7.8. ACTIVATING THE RPG II EDITOR (@RPG)

The RPG directive activates the RPG II editor (assuming that your system is configured with an RPG II editor) to create and maintain RPG II source modules. During an EDT session, you activate the RPG II editor by keying in the following directive:

```
@RPG
```

The help screen format is:



```
SYNTAX: @RPG [<PARAM-STRING>]
```

If you are not currently in an EDT session, to activate the RPG II editor you must first key in EDT (to activate EDT) and then follow it with the RPG directive:

```
EDTΔ@RPG
```

If you are already using the COBOL editor, end your COBOL editor session before activating the RPG II editor.

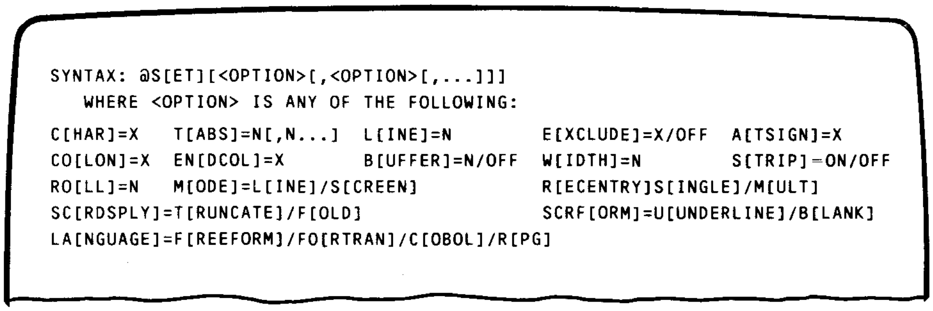
For further information about the RPG directive, see the RPG II editor user guide/programmer reference.

7.9. SPECIFYING PARAMETERS TO EDT (@SET)

The SET directive defines various parameters to EDT. Collectively, these parameters make up what we call your EDT environment. The format is:

```
@SET [CHAR=tab-character,TABS={columns}][,LINE=[length]]
     [                        {OFF    }]

   [,EXCLUDE={exclusion-character}][,ATSIGN=command-trigger]
   [         {OFF                }]

   [,COLON=range-separator][,ENDCOL=end-column][,BUFFER={record-size}][,CLEAR]
                                               [        {OFF        }]

   [,WIDTH=device-size][,STRIP={OFF}][,DISPLAY]
                       [       {ON }]

   [,SCRDSPLY {TRUNCATE}][,ROLL={15(if SCRDSPLY=TRUNCATE}]
   [          {FOLD    }][      {8(if SCRDSPLY=FOLD)    }]
                         [      {1-15                  }]

   [,MODE={LINE  }][RECENTRY={SINGLE}][,LANGUAGE={FREEFORM}]
   [      {SCREEN}][         {MULT  }][          {FORTRAN }]
                                      [          {COBOL   }]
                                      [          {RPG     }]

   [,SCRFORM={UNDERLINE}]
   [         {BLANK    }]
```

The help screen format is:



```
SYNTAX: @S[ET][<OPTION>[,<OPTION>[,...]]]
   WHERE <OPTION> IS ANY OF THE FOLLOWING:
C[HAR]=X      T[ABS]=N[,N...]  L[INE]=N        E[XCLUDE]=X/OFF  A[TSIGN]=X
CO[LON]=X     EN[DCOL]=X       B[UFFER]=N/OFF  W[IDTH]=N        S[TRIP]=ON/OFF
RO[LL]=N      M[ODE]=L[INE]/S[CREEN]           R[ECENTRY]S[INGLE]/M[ULT]
SC[RDSPLY]=T[RUNCATE]/F[OLD]                   SCRF[ORM]=U[UNDERLINE]/B[LANK]
LA[NGUAGE]=F[REEFORM]/FO[RTRAN]/C[OBOL]/R[PG]
```

where:

```
CHAR=tab-character,TABS={columns}
                        {OFF    }
```

Specifies the character EDT uses as the tab key and the columns it uses as tab stops. To use tabs, you must specify both parameters, CHAR and TABS.

You may specify CHAR= as any character on the workstation keyboard except your command trigger (@) and these predefined symbols: * & $ % ? [ ].

When specifying TABS, you must separate the column numbers where you want tab stops with commas. And you can specify a maximum of eight stops. Note that TABS=OFF clears previously set stops.

For example, CHAR=; assigns the semicolon as your tab key, while TABS=10,30 sets your tab stops at columns 10 and 30.

When you actually press your tab character while entering data, EDT does not move the cursor to the appropriate tab the way a typewriter does; however, in the future, whenever EDT displays that line on the workstation or prints a listing of it, it will be in the correct tab format you have designed.

For example, assume your tab character and columns are those we just specified in the previous example. If you key in

```
13.ØØØØ▷USER1;SPERRY UNIVAC;BLUE BELL, PA
```

in the future EDT will display line 13 as:

```
13.ØØØØ▷USER1          SPERRY UNIVAC          BLUE BELL, PA
```

If you want to skip an entry under the first tab column, you key in

```
13.ØØØØ▷USER1;;BLUE BELL, PA
```

in the future EDT will display line 13 as:

```
13.ØØØØ▷USER1                                 BLUE BELL, PA
```

NOTE:

The tab character is ignored during an @UPDATE process.

```
LINE=length
```

Specifies the maximum number of characters to be printed on a line at the workstation. The length may be from 1 to 2048 characters. For example, LINE=50 sets the maximum length for print lines to 50 characters. Note that if the length is omitted (LINE =), the length of the lines is set to 128 characters.

```
EXCLUDE={exclusion-character}
        {OFF                }
```

Specifies a character used to mask other characters. An exclusion character may be either a letter from A to Z or an integer from 0 to 9. For example, to change the first three characters on lines 1 through 4 in the following example, establish X as the exclusion character (EXCLUDE=X). Then change the masked characters with the command CHANGE 'XXXDATA' to 'EMPDATA'.

Lines 1 through 4 are changed to EMPDATA:

```
1.0000▷@SET EX=X
1.0000▷MSKDATA
2.0000▷LKTDATA
3.0000▷RAMDATA
4.0000▷PEHDATA
5.0000▷@CHANGE  'XXXDATA'  TO  'EMPDATA'  PRINT
1.0000▷EMPDATA
2.0000▷EMPDATA
3.0000▷EMPDATA
4.0000▷EMPDATA
```

Note that if EXCLUDE=OFF is specified, no exclusion character is assigned.

ATSIGN=command-trigger

Defines the character to be recognized as the current command trigger. It is used to change the preset command trigger (@) to another character. The command trigger may be set to any character found on the workstation keyboard except for these predefined symbols: * & % $? [and]. For example, ATSIGN=# changes the command trigger to #. Therefore, an executable EDT command would look like #COPY 1 to 5 instead of @COPY 1 to 5.

COLON=range-separator

Defines a character to be recognized as the column and line range separator. It is used to change the preset range separator (:) to another character. The range separator may be set to any character found on the workstation keyboard except for these predefined symbols: * & % $? [ ]. For example, COLON=– makes the hyphen the range separator. Therefore, a line range would now look like 20–30 instead of 20:30.

ENDCOL=end-column

Defines the end column for a search operation (i.e., a command using a search string). It is used to change the end-column indicator (]) preset by a previous FIND command to another character. The end-column indicator may be set to any character found on the workstation keyboard except for these predefined symbols: * & $ % ? [. For example, ENDCOL=! sets the end-column indicator to an exclamation point.

```
BUFFER={record-size}
       {OFF        }
```

Defines the maximum record size of a user record. The record size may be set in the range of 128 to 2048. The default is 128. BUFFER=OFF sets the buffer size to 128. Note that in order to change the record size, the work space must be closed. You close the work-space file via the DROP command.

```
CLEAR
```

Resets the error indicator.

```
WIDTH=device-size
```

Defines the output device size. The device size may be set in the range of 1 to 2048. If an odd number is specified, the device size is automatically rounded to the next even number. The default values are 120 for batch mode and 960 for interactive mode.

```
STRIP={OFF}
      {ON }
```

Indicates whether data lines are to be stripped of trailing spaces and X'00's. ON specifies that trailing spaces and X'00's are stripped. OFF specifies that they are to be retained.

```
DISPLAY
```

Displays the current status (values) of the SET command parameters. It's the same as using the SET command without any parameters.

```
SCRDSPLY={TRUNCATE}
         {FOLD    }
```

Specifies that you want to display or input either full-length character records (those containing 70 to 128 characters) or partial records (those containing 70 characters or less). Your specification for this parameter determines the record length that EDT accepts or displays when you issue:

- the EDT commands @BLOCK, @PRINT, or @ROLL; or

- the @CHECK=ON directive.

SCRDSPLY=TRUNCATE specifies that you want EDT to truncate all lines that you enter, or that it displays, to 70 characters or less. This specification allows you to enter or display more records per screen than SCRDSPLY=FOLD because SCRDSPLY=TRUNCATE causes EDT to accept or display one screen line per one source or data line.

SCRDSPLY=FOLD specifies that you don't want EDT to truncate lines you enter, or that it displays, to 70 characters or less. When SCRDSPLY=FOLD, EDT displays and accepts 128 characters. This specification reduces the number of records per screen that EDT can display because SCRDSPLY=FOLD causes EDT to accept or display two screen lines for every single source or data line.

```
ROLL=⎧15(if SCRDSPLY=TRUNCATE)⎫
     ⎨8(if SCRDSPLY=FOLD)     ⎬
     ⎩1-15                    ⎭
```

Specifies the number of lines EDT rolls forward when you issue an @ROLL screen command. See Section 8 for a description of the @ROLL screen command.

Anytime you change your specification for the SCRDSPLY parameter, EDT automatically changes the value for ROLL if you don't change it yourself. Because SCRDSPLY=TRUNCATE allows EDT to display twice as many source or data lines per screen as SCRDSPLY=FOLD does, your value for ROLL when SCRDSPLY=TRUNCATE can always be approximately twice as much as your value for ROLL when SCRDSPLY=FOLD.

```
MODE=⎧LINE  ⎫
     ⎩SCREEN⎭
```

Specifies whether EDT operates in line or screen mode. See Section 10 for detailed information about screen mode processing.

NOTE:

The following three parameters apply only if you specified that you want EDT to operate in screen mode (MODE=SCREEN). For information on specifying them, see Section 10.

```
LANGUAGE=⎧FREEFORM⎫
         ⎪FORTRAN ⎪
         ⎨COBOL   ⎬
         ⎩RPG     ⎭
```

Identifies the type of formatted screen that EDT presents for entering source code or data in screen mode.

```
RECENTRY=⎧SINGLE⎫
         ⎩MULT  ⎭
```

Identifies the number of lines you want to enter in one transmission.

```
SCRFORM=⎧UNDERLINE⎫
        ⎩BLANK    ⎭
```

Defines whether or not EDT displays underlines to indicate fields you can fill in.

It is not necessary to specify all the parameters with the SET directive. They may be specified alone, in combinations, or all at once. The SET directive may also be used without any parameters, in which case a listing of the values currently assigned to all the SET directive parameters is produced, as well as a display of the last used file string.

NOTE:

Do not separate parameters by spaces. When EDT encounters a space in the SET directive, it terminates the directive at that point.

Some typical examples of using the SET directive are:

```
@S C=/,T=12,24,36,48
```

Sets the tab key to a slash and tab columns to 12, 24, 36, and 48. Thus, everytime a slash is keyed in, the data following it is shifted to the next tab stop (12, 24, 36, or 48 in this case).

```
@S L=70
```

Sets the maximum line length to 70 characters. Therefore, only the first 70 characters keyed in on a line will be displayed on the workstation. If a line longer than 70 characters is keyed in, the line is truncated after the last full word inside that 70-character boundary.

```
@S A=/,CO=; ,EN=!
```

Sets the command trigger to a slash (/), the range separator to a semicolon (;), and the end column indicator to an exclamation point (!)

```
@S A=q, CO=i
```

Sets the command trigger to q. Note that the range separator is not reset to i because EDT detects a space before CO=i, which terminates the command.

```
@S M=S,LA=C,SC=F
```

Activates screen mode processing for you to enter COBOL source code with no source lines truncated.

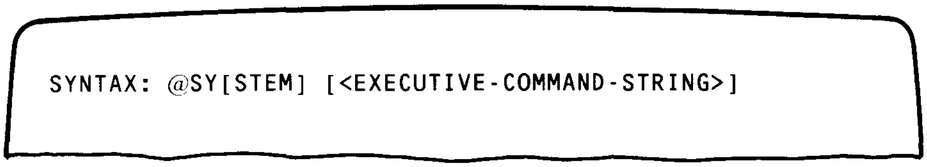
7.10. EXECUTING WORKSTATION COMMANDS DURING AN EDT SESSION (@SYSTEM)

The SYSTEM directive permits workstation commands to be issued during an EDT session, or it returns you temporarily to system mode. The main work file and any defined EDT procedure files are retained.

The format is:

```
@ SYSTEM [ workstation-command]
```

The help screen format is:



```
SYNTAX: @SY[STEM] [<EXECUTIVE-COMMAND-STRING>]
```

where:

`workstation-command`
: Specifies a workstation command (as defined in the interactive services commands and facilities user guide/programmer reference) that is to be keyed in during an EDT session.

If you specify @SYSTEM without a workstation command specified, EDT temporarily returns you to system mode and displays a message telling you to enter a system command. In response to this message, key in the system command you want to enter at the start-of-entry character (▷) and press the XMIT key. When you have no other system commands to enter, key in RESUME at the start-of-entry character and press the XMIT key. Your system then returns control to EDT, which positions the cursor at the next available line to accept your next command or data.

While your system processes the system commands you entered, you will see SYS MSG, for system message, displayed on the system state line. To see the message, press FUNCTION and SYSMODE simultaneously. You'll then see the message displayed on the top of your screen. Press the XMIT key after reading each message. When there are no more system messages, press the XMIT key once more and your system will reposition the cursor at the next available EDT line.

Some typical examples of the SYSTEM directive are:

@SY TELL
Enables you to interrupt your EDT session to send messages to the system console operator

@SY ASK
Enables you to interrupt your EDT session to ask questions of the system console operator

@SY
Returns you temporarily to system mode without releasing the work-space file

8. Screen Commands

8.1. INTRODUCTION

The EDT screen commands display screens that make EDT easier to use. Although each command displays a different kind of screen to help you perform a different function, the screen commands have one thing in common: they all make EDT easier to use because they reduce the need for you to interrupt your EDT session to get information about your system, in general, and about EDT, specifically. In short, they help you conduct an entire EDT session without leaving your workstation screen.

(The only screen command that doesn't display a screen is the RESTORE command. This command's function is to return you to the point in your EDT session where you issued another screen command. In this way, it too makes EDT easier to use.)

Although these commands are called screen commands, there's no reason for you to restrict their use to screen mode only. You can use all the screen commands in line or screen mode; in both modes, they provide the same functionality for you. Whatever mode you're in when you issue a screen command is the mode EDT returns you to after it processes the screen command.

Each screen command has a function key equivalent. When pressed, the function key performs the exact same function as the command. See Section 9 for a description of function key use.

Like all other EDT commands, you must precede all screen commands with an at sign (@). Once EDT processes the screen command you've entered, it displays a screen. Before pressing the XMIT key on any of these screens, be sure to move the cursor to the bottom rightmost position on the screen to ensure that EDT transmits all the information on the screen to the work-space file.

The following subsections describe the EDT screen commands. They present the commands in alphabetical order for easy reference. For each command, we give a description of the command, the format for entering it, detailed information about how to use the command, and an example of its use. For your convenience, Table 8–1 summarizes the screen commands.

NOTE:

With the exception of the PROMPT screen command, all screen commands operate independently. Therefore, you can't combine any of them with any other EDT commands, nor can you use modifiers with any screen command.

Table 8-1. Screen Commands

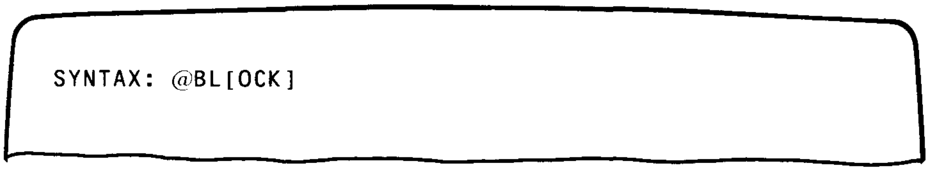
Command	Function
BLOCK	Displays a freeform screen that allows you to switch to block mode for entering multiple commands or data.
HELP	Displays help screens for any EDT error messages.
PARAMS	Displays a screen showing the parameters of the @SET directive (those that make up your EDT environment).
PROMPT	Displays the EDT command menu screen and help screens for any of the EDT commands (meaning EDT commands, modifiers, directives, procedure file commands, variables, and screen commands.)
RESTORE	Returns you to the point in your EDT session where you originally entered a screen command.
ROLL	Displays freeform screens, showing the EDT work-space file, where you can update lines or simply view them.

8.2. ENTERING MULTIPLE COMMANDS AND DATA (@BLOCK)

The BLOCK screen command lets you switch to block mode from line or screen mode. In block mode, EDT displays a freeform screen so you can enter multiple EDT commands or data. The format of the BLOCK command is:

```
@BLOCK
```

The help screen format is:



```
SYNTAX: @BL[OCK]
```

The function key equivalent for the BLOCK command is F4. Pressing F4 is the same as keying in the @BLOCK command.

In block mode, you can enter one or more lines and can combine commands and data in one transmission. EDT processes the lines in the order you entered them and doesn't display another freeform screen until it processes all the lines in one transmission.

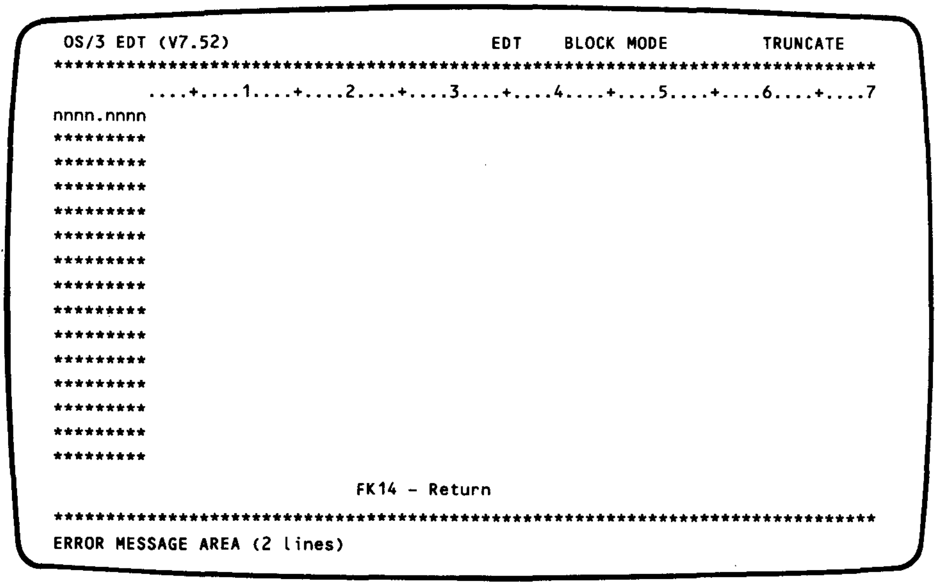
EDT places data in the work space sequentially from the current line number, places embedded blank lines in the work space as blank lines, and ignores trailing blank lines.

When working in block mode, you cannot enter any other screen commands except @RESTORE and you can't use the directives @COBOL, @RPG, or @EFP. In addition, you can't use @BLOCK in an active procedure file. That is, if you execute a procedure file that contains @BLOCK, you will encounter errors.

The maximum number of characters you can enter on one line of the block mode freeform screen depends on your specification for the SCRDSPLY parameter of the @SET directive. If SCRDSPLY=TRUNCATE, you can enter a maximum of 70 characters on one line; if SCRDSPLY=FOLD, you can enter a maximum of 128 characters.

When you enter block mode, the first line of the freeform screen contains the current line number. The rest of the lines are preceded by asterisks that EDT protects and removes when you press the XMIT key. You terminate block mode by pressing the F14 function key. EDT then returns you to the mode you were in when you issued the @BLOCK command, at the point in your EDT session where you originally entered the @BLOCK command.

The following is the block mode freeform screen that EDT displays when you enter block mode and your value for SCRDSPLY of the @SET directive is TRUNCATE:



```
 OS/3 EDT (V7.52)                        EDT    BLOCK MODE         TRUNCATE
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
nnnn.nnnn
*********
*********
*********
*********
*********
*********
*********
*********
*********
*********
*********
*********
*********
*********
                              FK14 - Return
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

On this one screen, you could, for example, issue an @CHANGE command to change a string already in your work-space file to a string you provide with the command. You could also issue an @INSERT command to insert a totally new string. Before you transmit those commands, you could issue an @PRINT command to have EDT display the changes on your workstation screen and an @LIST command to have EDT list its work-space contents on your printer. Then, in one transmission, EDT would process those commands one after another. To move the cursor to the next line without transmitting them, press the TAB FORWARD key. Your entries could look something like the following:

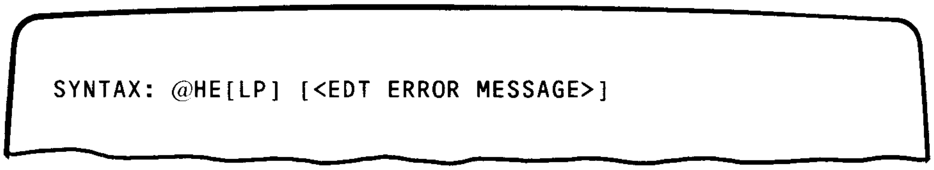
```
@CHANGE 'BOB' TO 'ROBERT'
@INSERT 'ABC'
@PRINT
@LIST
```

8.3. REQUESTING HELP WITH EDT ERROR MESSAGES (@HELP)

The HELP screen command lets you display a help screen for any EDT error message. The format of the HELP command is:

```
@HELP [error message code]
```

The help screen format is:



```
SYNTAX: @HE[LP] [<EDT ERROR MESSAGE>]
```

where:

`error message code`

Specifies the error message prefix and number that precedes every error message. For example, in error message

```
ED016  INVALID LINE SET COMMAND,
```

ED016 is the error message code.

The function key equivalent for the HELP command is F6. Pressing F6 is the same as keying in the @HELP command with no error message code.

If you enter the HELP command with no error message code, EDT displays a help screen for the last EDT error encountered. On the other hand, if you specify an error message code on the HELP command, EDT shows the help screen only for that specific message.

The error message help screen:

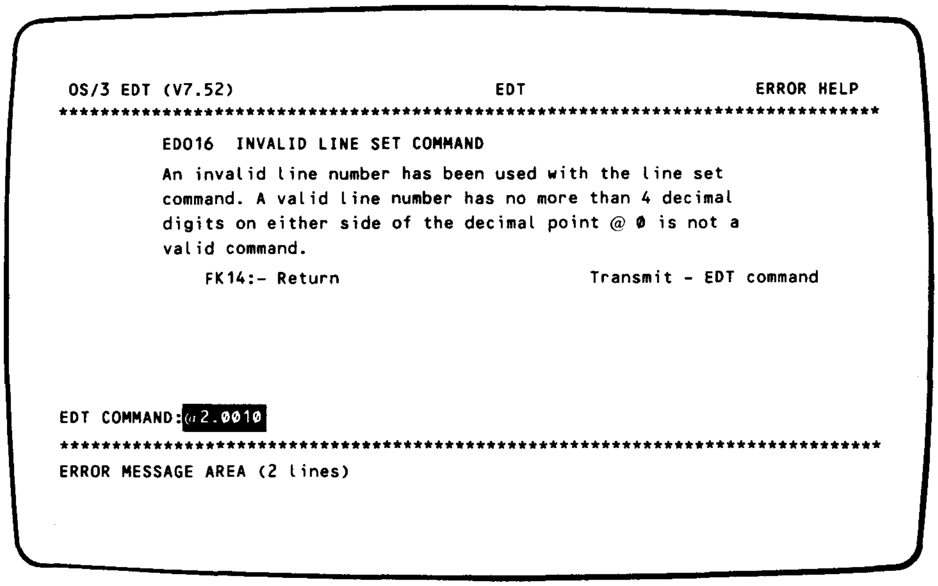
- shows the error message prefix, number, and text;
- gives an explanation of the error; and
- includes a line for you to enter an EDT command to fix the error or reissue an improperly entered command, if possible.

After EDT processes the command, EDT returns you to the point where you issued the HELP command. If you don't want to enter an EDT command, simply press the XMIT key or the F14 function key to return to the point where you encountered the error and entered the HELP command.

Note that the EDT HELP command doesn't display help screens for messages other than EDT error messages. To get help for error messages from EDT's subprocessors (the COBOL and RPG II editors and the error file processor), or from other system components, enter system mode and issue an interactive services HELP command. See the @SYSTEM directive for the procedure for temporarily entering system mode during an EDT session. The interactive services commands and facilities manual gives a description of the interactive services HELP command.

In some cases, EDT can't display a help screen for a message because EDT's error help screen file doesn't contain a help screen for that error message. In these cases, see the system messages manual for a description of your error and the corrective action you should take.

The following is a sample EDT error message help screen:



```
 OS/3 EDT (V7.52)                       EDT                      ERROR HELP
*************************************************************************************
          ED016  INVALID LINE SET COMMAND
          An invalid line number has been used with the line set
          command. A valid line number has no more than 4 decimal
          digits on either side of the decimal point @ Ø is not a
          valid command.
              FK14:- Return                          Transmit - EDT command




EDT COMMAND:@2.ØØ1Ø
*************************************************************************************
ERROR MESSAGE AREA (2 lines)
```

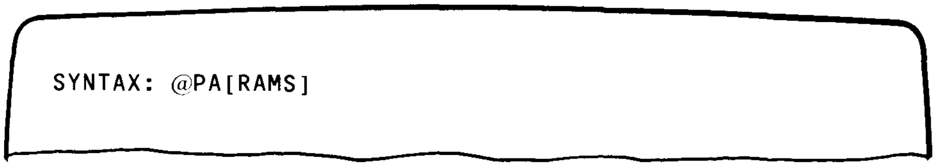
To correct the error, you could reissue the @ command with a valid line number. For example, you'd specify a line number of no more than four digits on either side of the decimal point, say, 2.0010. (Your entry is shown as white letters on a black background on the sample screen.) EDT then sets the current work-space line number to that number and returns you to the point in your EDT session where it reported error EDO16.

8.4. REVIEWING THE EDT ENVIRONMENT PARAMETERS (@PARAMS)

The PARAMS screen command lets you request a screen showing all your current values for the parameters of the @SET directive. These parameters collectively make up what we call your EDT environment. The format of the PARAMS command is:

```
@PARAMS
```

The help screen format is:



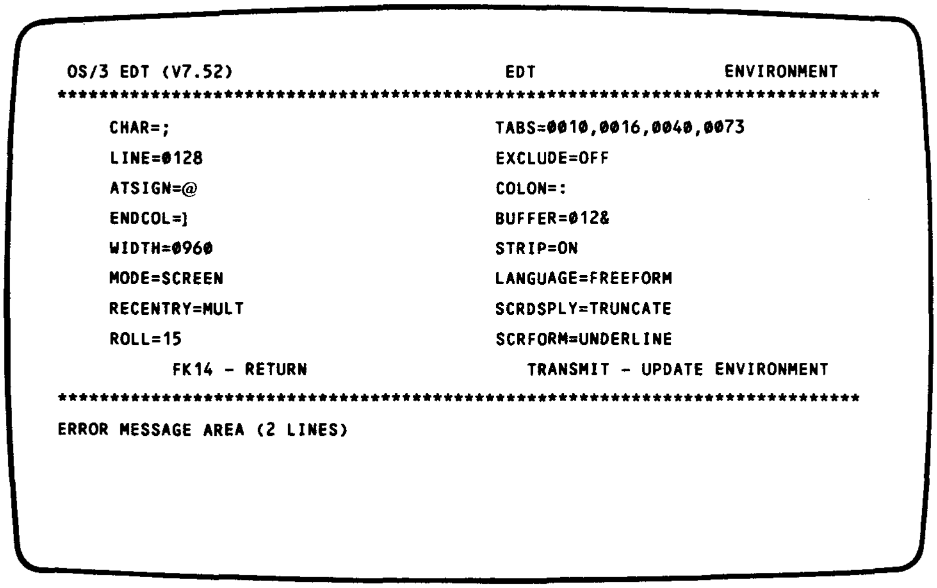
```
SYNTAX: @PA[RAMS]
```

The function key equivalent of the PARAMS command is F3. Pressing F3 is the same as keying in the @PARAMS command.

When EDT displays the environment screen, you can update any number of the @SET parameters. You do so by simply moving the cursor to the values you want to change, typing the new values over the current values, moving the cursor to the bottom rightmost position on the screen, and pressing the XMIT key. EDT then updates the @SET parameters and returns you to the point in your EDT session where you issued the @PARAMS command. The only difference in your EDT session, provided you changed any of the parameters, is that your environment will be different in whatever way you indicated through the changes to the @SET parameters. (Of course, you could also directly issue an @SET directive to change EDT environment parameters. But the PARAMS command lets you review your parameters first, then you can change them if you want to.)

After reviewing your parameters, if you don't want to change any, simply press the F14 function key and EDT returns you to the point in your EDT session where you issued the PARAMS command. In these cases, your environment doesn't change.

The following is a typical environment screen that EDT would display in response to a PARAMS command:



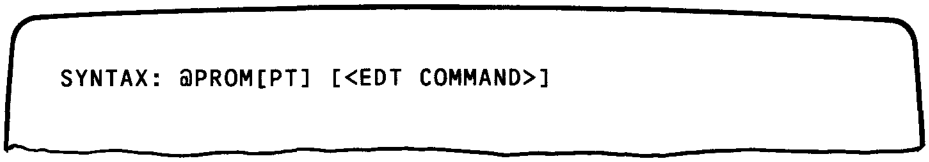
```
 OS/3 EDT (V7.52)                          EDT                    ENVIRONMENT
********************************************************************************
     CHAR=;                                TABS=ØØ1Ø,ØØ16,ØØ4Ø,ØØ73
     LINE=Ø128                             EXCLUDE=OFF
     ATSIGN=@                              COLON=:
     ENDCOL=]                              BUFFER=Ø12&
     WIDTH=Ø96Ø                            STRIP=ON
     MODE=SCREEN                           LANGUAGE=FREEFORM
     RECENTRY=MULT                         SCRDSPLY=TRUNCATE
     ROLL=15                               SCRFORM=UNDERLINE
           FK14 - RETURN                      TRANSMIT - UPDATE ENVIRONMENT
******************************************************************************
ERROR MESSAGE AREA (2 LINES)
```

8.5. DISPLAYING THE EDT COMMANDS AND THEIR HELP SCREENS (@PROMPT)

The PROMPT screen command lets you display a menu screen showing all the EDT commands (meaning the EDT commands, modifiers, directives, procedure file commands, variables, screen commands, and function keys that you can use in EDT). It also lets you request a help screen for any of these commands. The format of the PROMPT command is:

```
@PROMPT∆[edt command]
```

The help screen format is:



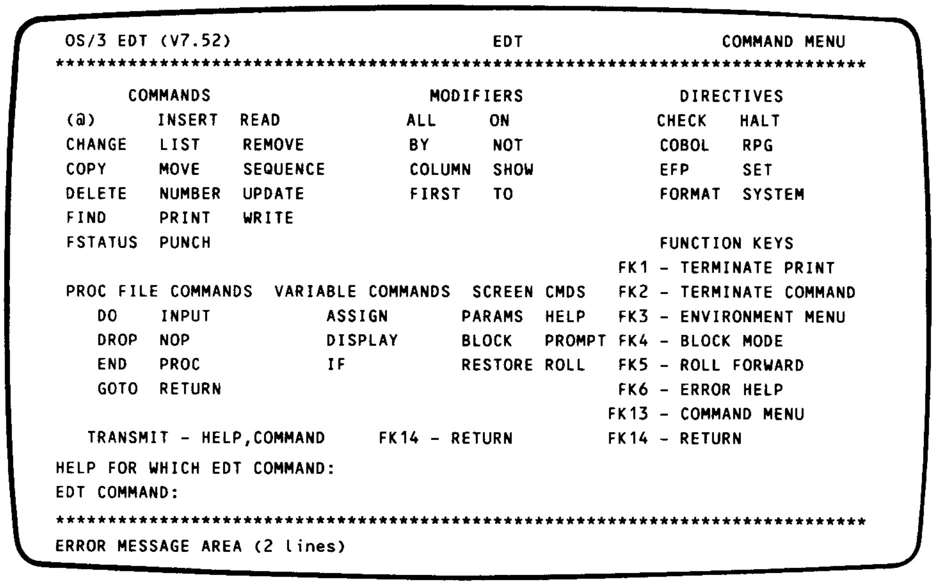
```
SYNTAX: @PROM[PT] [<EDT COMMAND>]
```

where:

edt command
: Specifies that you want to see a help screen for a specific EDT command. For example, @PROMPT NUMBER shows a help screen for the EDT NUMBER command. When requesting a specific help screen, you can use the full name of the EDT command or its abbreviation.

The function key equivalent for the PROMPT command is F13. Pressing F13 is the same as keying in an @PROMPT command with no EDT command specified.

With no EDT command specified on the PROMPT command, EDT displays the following EDT command menu screen:



```
 OS/3 EDT (V7.52)                         EDT                   COMMAND MENU
******************************************************************************
       COMMANDS                     MODIFIERS               DIRECTIVES
 (@)      INSERT  READ            ALL     ON              CHECK   HALT
 CHANGE   LIST    REMOVE          BY      NOT             COBOL   RPG
 COPY     MOVE    SEQUENCE        COLUMN  SHOW            EFP     SET
 DELETE   NUMBER  UPDATE          FIRST   TO              FORMAT  SYSTEM
 FIND     PRINT   WRITE
 FSTATUS  PUNCH                                           FUNCTION KEYS
                                                      FK1 - TERMINATE PRINT
 PROC FILE COMMANDS  VARIABLE COMMANDS  SCREEN CMDS   FK2 - TERMINATE COMMAND
    DO    INPUT           ASSIGN       PARAMS  HELP   FK3 - ENVIRONMENT MENU
    DROP  NOP             DISPLAY      BLOCK   PROMPT FK4 - BLOCK MODE
    END   PROC            IF           RESTORE ROLL   FK5 - ROLL FORWARD
    GOTO  RETURN                                      FK6 - ERROR HELP
                                                     FK13 - COMMAND MENU
   TRANSMIT - HELP,COMMAND     FK14 - RETURN         FK14 - RETURN
HELP FOR WHICH EDT COMMAND:
EDT COMMAND:
******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

From this command menu screen, you can do one of the following:

- enter an EDT command by keying in the command after the EDT COMMAND: message and pressing the XMIT key;

- request to see a help screen for a specific EDT command by keying in the command after the message, HELP FOR WHICH EDT COMMAND:, and pressing the XMIT key (note that you can also display a help screen by including a specific command in your PROMPT command); or

- return to your EDT session at the point where you originally entered the PROMPT command by pressing the F14 function key or by simply pressing the XMIT key.

NOTE:

Be sure that you select only one of the preceding actions from the command menu screen. If you enter both an EDT command and a request for a specific command help screen, EDT will not process the command you enter on the EDT COMMAND line.

EDT command help screens show the syntax for the command and, if possible, a command entry line where you can enter any EDT command. The help screens give short, concise descriptions of the commands. For more detailed information about a command, see the appropriate command description in this manual.

The EDT help screen command formats differ slightly from the formats we present in this manual. These differences exist because the workstation screen (or one of the display devices that supports EDT) can't display all of the characters in the formats we present in this manual. Although the syntax appears to be different, you should specify the EDT commands according to the statement of conventions we present in Section 1. Specifically, the conventions in this manual differ from those in the help screens in the ways the following table describes:

CONVENTIONS USED IN THIS MANUAL	CONVENTIONS USED IN COMMAND HELP SCREENS
Underscoring (_) indicates required letters in the abbreviations for commands and parameters. `@MOVE`	Brackets ([ ]) around non-required letters in commands and parameters indicate the letters are optional. `@M[OVE]`
Apostrophes (') enclose search-strings, change-strings, sequence-strings, and strings.	Less-than and greater-than signs enclose all strings. `<SEARCH-STRING>`
Shaded values NO indicate default values. YES	No default values are shown.

Brackets large enough to surround a parameter indicate the parameter is optional. $\left[\texttt{,DEVICE=} \begin{matrix}\texttt{did}\\ \texttt{DISK}\\ \texttt{DISKETTE}\end{matrix}\right]$	Brackets that surround only the line on which the parameter name lies indicate the parameter is optional.
	`            DID`
	`[,DEVICE= DISK     ]`
	`            DISKETTE`

All command formats include the optional string multiplication factor (*n).	No command help screens include the optional string multiplication factor.

Choices are shown by braces ({}) enclosing a number of entries from which you must make a choice of one.
$\begin{Bmatrix}\texttt{did}\\ \texttt{DISK}\\ \texttt{DISKETTE}\end{Bmatrix} \begin{Bmatrix}\texttt{data}\\ \texttt{command}\end{Bmatrix}$
	Choices may be shown in one of several ways:
	■ More than one format indicates you must choose one of the formats;
	■ Slashes (/) between entries indicate you must choose one of the entries shown on either side of the slashes. `<DATA>/<COMMAND>`
	■ Less-than and equal-than signs indicate you must choose one of several values.
	`  DID`
	`< DISK     >`
	`  DISKETTE`

Uppercase letters indicate the command, modifier, or parameter must be keyed in exactly as shown; lowercase letters indicate user-defined variables. `@FIND 'search-string'`	All entries are uppercase, whether constant or variable. `@FIND 'SEARCH-STRING'`

From the command help screens, you can do one of the following:

- Enter the EDT command explained in the screen by entering your command trigger and the command on the EDT command entry line and pressing the XMIT key

- Enter an EDT command different from the one explained in the screen by the same method just described

- Request that EDT display additional help for this command (such as additional syntax or definitions of the terms in the format) by pressing the F13 function key. From an additional prompt help screen, you can:

  - request to see a previously shown additional help screen by pressing FK12;

  - request to see a subsequent additional help screen by pressing F13 again;

  - enter an EDT command on the EDT command entry line and press the XMIT key; or

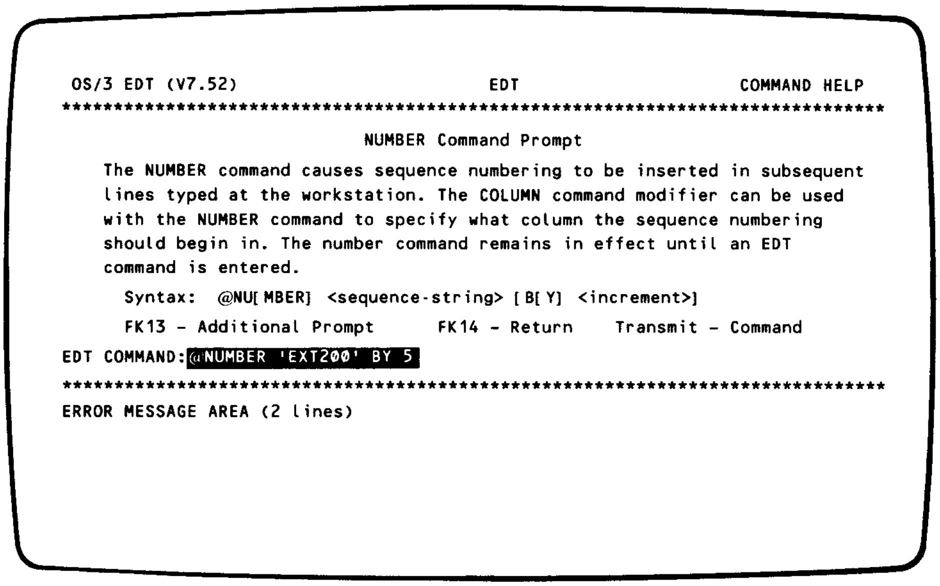
  - press F14 to return to the point where you originally requested help for the command.

  Each command help screen to which EDT can supply additional help displays your choice of actions at the bottom of the screen. If the choice FK13–ADDITIONAL PROMPT does not appear on the screen, EDT can supply no more additional help for the command. At that point, you can still perform any of the choices the screen does display. You perform the action exactly as we've just described.

- Return to your EDT session at the point where you originally entered the PROMPT command by pressing the F14 function key

The following example illustrates a typical use of the PROMPT command:

You want to review the EDT commands, so you issue an @PROMPT command with no specific command after it. EDT displays the command menu screen and you decide you want to get help for the NUMBER command. You, therefore, key in NUMBER after the message, HELP FOR WHICH EDT COMMAND: (see the command menu screen shown previously). EDT then displays the help screen for the NUMBER command, which looks something like this:



```
OS/3 EDT (V7.52)                        EDT                      COMMAND HELP
********************************************************************************
                              NUMBER Command Prompt
    The NUMBER command causes sequence numbering to be inserted in subsequent
    lines typed at the workstation. The COLUMN command modifier can be used
    with the NUMBER command to specify what column the sequence numbering
    should begin in. The number command remains in effect until an EDT
    command is entered.
      Syntax:  @NU[MBER] <sequence-string> [B[Y] <increment>]
      FK13 - Additional Prompt      FK14 - Return    Transmit - Command
EDT COMMAND:@NUMBER 'EXT200' BY 5
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

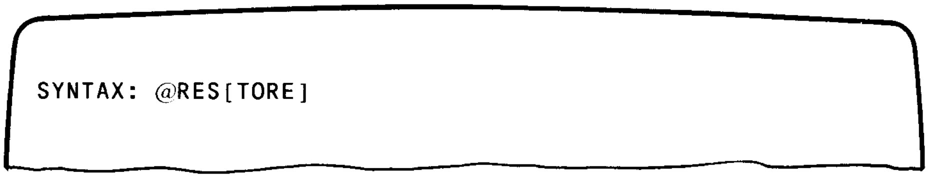
After reading this screen, you want to issue the NUMBER command. So you key in the NUMBER command as shown in the format and press the XMIT key. (Your keyin is shown as white lettering on a black background on the sample screen.) EDT then processes the NUMBER command and returns you to the point in your EDT session at which you originally entered the PROMPT command.

8.6. RETURNING TO YOUR EDT SESSION FROM A SCREEN COMMAND (@RESTORE)

The RESTORE screen command returns you to the point where you originally entered a screen command. The format of the RESTORE command is:

```
@RESTORE
```

The help screen format is:



```
SYNTAX: @RES[TORE]
```

The function key equivalent of the RESTORE command is F14. Pressing F14 is the same as keying in the @RESTORE command.

The following example illustrates a typical use of the RESTORE command:

Let's say you want to display the contents of the EDT work space on your workstation screen but you can't remember whether you need the LIST command or the PRINT command. You, therefore, issue an @PROMPT command to see all the EDT commands. After reading the list of EDT commands, you request the help screen for the LIST command. The help screen tells you that the LIST command prints the work-space contents on your printer, so you conclude that you need the PRINT command. At this point, you issue an @RESTORE command to return to the point in your EDT session where you first issued the PROMPT command. Then you enter an @PRINT command and EDT displays the contents of its work space on your workstation screen.

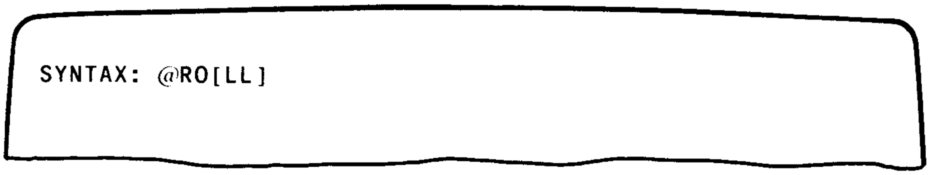
RO

8.7. VIEWING AND UPDATING LINES (@ROLL)

The ROLL screen command lets you roll through EDT's work-space file displaying lines and, if you want, updating or correcting them. The format of the ROLL command is:

```
@ROLL
```

The help screen format is:



```
SYNTAX: @RO[LL]
```

The function key equivalent of the ROLL command is F5. Pressing F5 is the same as keying in the @ROLL command.

When you enter an @ROLL command, EDT displays its work-space contents on freeform screens that contain the number of lines you specified for the ROLL parameter on the @SET directive. The number of characters across one freeform screen depends on your specification for the SCRDSPLY parameter of the @SET directive.

EDT displays the lines starting with the current line number. When it reaches the end of the work-space file, it starts displaying lines, beginning with line 1 of the work-space file. The roll function continues until you tell EDT to terminate it.

When EDT displays each freeform screen, you have three options:

- Update the currently displayed lines.

  You do this by:

  - moving the cursor to the items you want to change, insert, or delete;
  - making those updates using keyboard keys (not EDT commands);
  - moving the cursor to the bottom rightmost position on the screen; and
  - pressing the XMIT key to transmit the updated lines to the EDT work space.

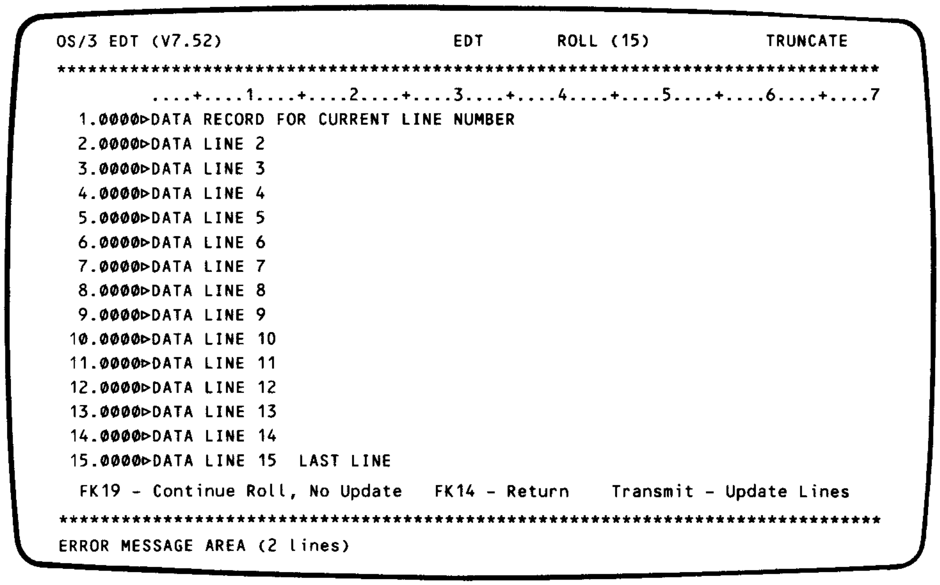
  EDT then displays the next set of lines.

- Roll to the next freeform screen without changing the currently displayed lines by pressing the F19 function key.

- Terminate the roll function by pressing the F14 function key.

  This option returns you to the point in your EDT session where you originally entered the ROLL command.

The following is a sample freeform screen that EDT shows in response to a ROLL command:



```
OS/3 EDT (V7.52)                    EDT       ROLL (15)         TRUNCATE
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  1.0000▷DATA RECORD FOR CURRENT LINE NUMBER
  2.0000▷DATA LINE 2
  3.0000▷DATA LINE 3
  4.0000▷DATA LINE 4
  5.0000▷DATA LINE 5
  6.0000▷DATA LINE 6
  7.0000▷DATA LINE 7
  8.0000▷DATA LINE 8
  9.0000▷DATA LINE 9
 10.0000▷DATA LINE 10
 11.0000▷DATA LINE 11
 12.0000▷DATA LINE 12
 13.0000▷DATA LINE 13
 14.0000▷DATA LINE 14
 15.0000▷DATA LINE 15  LAST LINE
  FK19 - Continue Roll, No Update   FK14 - Return    Transmit - Update Lines
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Because of the way this user has specified the SCRDSPLY and ROLL parameters of the @SET directive, EDT displays 15 truncated lines at a time.

9. Function Keys

The following function keys may provide some general assistance while using EDT:

- Function key 1 (F1)

  Shuts off the display of lines to the workstation while the current command continues to process. F1 is the same as F18.

- Function key 2 (F2)

  Terminates the current command.

- Function key 3 (F3)

  Displays a screen showing the parameters of the @SET directive, or those that make up your EDT environment. F3 is the same as issuing the PARAMS screen command (see Section 8).

- Function key 4 (F4)

  Displays a freeform screen through which you can switch to block mode for entering multiple commands or data. F4 is the same as issuing the BLOCK screen command (see Section 8).

- Function key 5 (F5)

  Displays freeform screens, showing the EDT work-space file, where you can update lines or simply view them. F5 is the same as issuing the ROLL screen command (see Section 8).

- Function key 6 (F6)

  Displays help screens for any EDT error messages. F6 is the same as issuing the HELP screen command (see Section 8).

- Function key 12 (F12)

  Lets you request a previously displayed additional help screen for a specific command when that command requires several help screens to fully describe it.

- Function key 13 (F13)

  Displays the EDT command menu screen and help screens for any of the EDT commands (meaning EDT commands, modifiers, directives, procedure file commands, variables, and screen commands). F13 is the same as issuing the PROMPT screen command (see Section 8). From a help screen, F13 also lets you see subsequent help screens needed to fully explain the command.

- Function key 14 (F14)

  Returns you to the point in your EDT session where you originally entered a screen command. F14 is the same as issuing the RESTORE screen command (see Section 8).

- Function key 15 (F15)

  Recognized as EOF indicator. Halts an EDT session run in batch mode or produces an error message in interactive mode.

- Function key 18 (F18)

  Shuts off the display of lines to the workstation while the current command continues to process. F18 is the same as F1.

- Function key 19 (F19)

  This is a system key that may be used to continue the output of data to the workstation once the workstation screen becomes full.

The function keys 7–11, 16, 17, and 20–22 are invalid while using EDT. If pressed, EDT terminates the current command and displays an error message. The cursor returns to the next available work-space line for your next entry.

When using EDT screens displayed by screen commands, the only function keys available to you are those displayed on the bottom of the screens.

When using screen-mode screens, all the valid function keys described in this section are available to you.

For more information on how to use these and other system function keys, see the interactive services commands and facilities user guide/programmer reference.

10. Screen Mode Processing

10.1. INTRODUCTION

Screen mode processing is an EDT facility that lets you enter source code or data, as easily as possible, from your workstation. In addition, you can use any of the EDT commands. Primarily, there are two major ways that EDT screen mode allows easier source or data entry.

First, it allows you to enter up to 14 lines in one transmission. Depending on whether you want to enter single or multiple lines in one transmission, either: (1) EDT displays the contents of its work space as you transmit lines and scrolls the single lines up as it fills the screen; or (2) you enter up to 14 lines at a time before you transmit them.

Second, screen mode provides formatted screens for entering source code or data. EDT supplies COBOL, RPG II, or FORTRAN IV screens. These screens identify the programming fields for whatever language you want to use.

Also, screen mode provides a freeform screen where you can enter ASSEMBLER source code or uniform data. Although the OS/3 component, screen format services, lets you build screens to input data within a program, EDT's screen mode lets you use screens to create data files independent of any program. Therefore, a practical application of screen mode would be for your data entry personnel to use the freeform screens to create or update name and address files, customer name and account number files, etc. You could then use any of these data files within any program you write.

If you are a COBOL or RPG II programmer, you may have used, or considered using, the EDT subeditors (the COBOL editor or the RPG II editor). In addition to providing formatted screens, these subeditors also perform syntax checking on your source code. For creating a COBOL or RPG II source module, then, you may prefer to use the language's subeditor. However, for updating, if the updates to your source code aren't extensive and you feel you don't need the syntax checking, you can use screen mode to make updates and changes very quickly. By using screen mode, you don't have to call up the subeditors for simple updates.

At this point, you should take note that processing in screen mode is not the same as using the EDT screen commands. The EDT screen commands operate in both line and screen mode and provide the same functions in both. Whatever mode you're in when you issue a screen command is the mode EDT returns you to after it processes the screen command you entered.

Also, when working in screen mode, you have all of EDT's capabilities. In fact, all the screens EDT displays in screen mode include an EDT command entry line. From this line (one on each screen), you can enter any of the EDT commands, except the UPDATE command. Screen mode uses the ROLL screen command for updating instead of the UPDATE command.

To use screen mode, you first activate EDT as we describe in Section 2. Then, to tell EDT you want to operate in screen mode, you use the MODE parameter of the @SET directive. Before you transmit the @SET directive, however, you can also use several other parameters to tell EDT how you want to enter your source code or data.

By choosing different values for these parameters of the @SET directive and combining them with the MODE parameter, you tell EDT what type of screen to display. This screen and the ones that follow it make entering your source code or data easier.

See 10.2 for details about the values you can choose for these parameters and examples of the screens you'll see as a result of combining the different values for the parameters. See 10.3 for details on how you use screen mode to enter source code or data. Finally, 10.4 gives a sample screen mode session and Appendix G shows several screen mode sessions.

You can use EDT screen mode in an interactive environment only. That is, you must be operating from your console/workstation in workstation mode or from a workstation. In addition, your workstation screen must be a 24 x 80 screen.

10.2. DEFINING YOUR SCREENS

As we mentioned in 10.1, you use a combination of parameter values for the @SET directive to tell EDT how you want to enter your source code or data in screen mode.

Specifically, the @SET directive parameters that you use to define your screens are: SCRDSPLY, MODE, RECENTRY, LANGUAGE, and SCRFORM. Each of these parameters allows you to define a different characteristic of the type of screens you want to use while entering your source code or data. Once you transmit the @SET directive with specifications for these five parameters (or accept the default values for them), EDT immediately displays the first screen of the type you've defined and you're in screen mode.

The following subsections (10.2.1 through 10.2.5) give your choices for each of these parameters and details about the values you can choose. Subsection 10.2.6 talks about the type of screens you'll see as a result of combining different values for the parameters and gives some examples.

No matter how you specify the parameters for the @SET directive, there are certain areas common to all the screens you'll see when operating in screen mode.

The top of the screen always shows the EDT version number and your values for the LANGUAGE, RECENTRY, and SCRDSPLY parameters. It also shows the screen column numbers.

The middle of the screen is where you enter your source code or data. It always includes an EDT command entry line for entering any EDT command (except the UPDATE command).

The bottom of every screen is reserved for displaying error messages you may receive from EDT or your system while operating in screen mode.

10.2.1. Defining the Character Length of Your Records (SCRDSPLY)

The screen display parameter (SCRDSPLY) tells EDT whether you want to display and input full-length character records (those containing up to 128 characters) or partial character records (those containing 70 characters or less).

The format of the SCRDSPLY parameter is:

$$\left[\underline{\text{SC}}\text{RDSPLY=}\begin{Bmatrix}\underline{\text{T}}\text{RUNCATE}\\ \underline{\text{F}}\text{OLD}\end{Bmatrix}\right]$$

SCRDSPLY=TRUNCATE specifies that you want EDT to truncate all lines you enter (or that it displays) to 70 characters or less. This specification allows you to enter or display more records per screen than SCRDSPLY=FOLD because SCRDSPLY=TRUNCATE causes EDT to accept or display one screen line per one source or data line.

When SCRDSPLY=TRUNCATE and RECENTRY=SINGLE, your screen's display area shows a 70-column display no matter what you specify for the LANGUAGE parameter.

SCRDSPLY=FOLD specifies that you don't want EDT to truncate lines you enter (or that it displays) to 70 characters or less. When SCRDSPLY=FOLD, EDT displays and accepts up to 128 characters. This specification reduces the number of records per screen that EDT can display because SCRDSPLY=FOLD causes EDT to accept or display two lines for every single source or data line.

When SCRDSPLY=FOLD and RECENTRY=SINGLE, your screens' display area shows a 128-column display no matter what you specify for the LANGUAGE parameter.

NOTE:

Regardless of your specification for the SCRDSPLY parameter, EDT processes 128-character records in screen mode. Also, EDT places the records in its work space with or without trailing spaces and X'00's, depending on your specification for the STRIP parameter of the @SET directive. In other words, if your records are less than 128 characters long, EDT pads them with spaces or strips them of trailing spaces and X'00's (depending on your specification for STRIP) before placing your records in the EDT work space.

10.2.2. Defining Your EDT Mode of Operation (MODE)

The mode of operation parameter (MODE) tells EDT that you want to enter screen mode. The format of the MODE parameter is:

```
MODE={LINE  }
     {SCREEN}
```

MODE=LINE alows you to operate only in line mode.

MODE=SCREEN allows you to operate only in screen mode. Of the five parameters that let you define your screens, this parameter is the only one you must specify (MODE=SCREEN) to enter screen mode. Therefore, it isn't an optional parameter for screen mode processing.

10.2.3. Defining the Number of Record Entries in One Transmission (RECENTRY)

The record entry parameter (RECENTRY) defines the number of records or lines you may enter before you press the XMIT key. The format of the RECENTRY parameter is:

```
[RECENTRY={SINGLE}]
[         {MULT  }]
```

RECENTRY=SINGLE specifies that you want to enter one record at a time, pressing the XMIT key after each one. If you specify RECENTRY=SINGLE, EDT displays a screen divided into four areas (see 10.2.6. for some examples):

- The environment area, which is common to all screen-mode screens
- The error message area, which is also common to all screen-mode screens
- A source code or data entry line, where you can enter one line at a time, and an EDT command entry line, where you can enter one EDT command at a time

From this section of the screen, you can enter both source code or data and an EDT command. If entering both, EDT places the source code or data in its work space first, then processes the EDT command. If you enter neither, EDT places a blank line in its work space after you press the XMIT key.

EDT precedes the entry line with its current work-space line number. Your specification for the LANGUAGE parameter determines the format of the source entry line.

- A display area that shows the source code or data you entered in the entry line. When you enter a line from the entry line, EDT displays it in the last line of the top (or display) portion of the screen before you enter the next line. When it fills the display area, EDT scrolls the lines up to make room for the rest of the lines you enter. EDT protects the display area; therefore, you can't enter any source code or data directly to it.

  EDT always displays your lines in set columns that are unrelated to the language format you used to enter your source code or data. When SCRDSPLY=TRUNCATE and RECENTRY=SINGLE, your screens' display areas show a 70-column display no matter what you specify for the LANGUAGE parameter. When SCRDSPLY=FOLD and RECENTRY=SINGLE, your screens' display areas show a 128-column display area, no matter what you specify for LANGUAGE.

RECENTRY=MULT specifies that you want to enter from 1 to 14 records or lines in one transmission. If you specify RECENTRY=MULT, EDT displays a screen divided into three areas (see 10.2.6 for some examples):

- The environment area, which is common to all screen-mode screens

- The error message area, which is also common to all screen-mode screens

- The source code or data entry area, which includes 14 lines where you can enter source code or data, and one EDT command entry line, where you can enter one EDT command at a time

  From this section of the screen, you can enter both source code or data and an EDT command. If entering both, EDT places the source code or data in its work space first, then processes the EDT command.

  EDT precedes the first entry line with its current work-space line number. The format of the entry lines is determined by your specification for the LANGUAGE parameter.

  EDT places embedded blank lines in its work space as blank lines; EDT doesn't place trailing blank lines in its work space.

10.2.4. Defining the Language Format of Your Screens (LANGUAGE)

The language parameter (LANGUAGE) tells EDT which type of formatted screen to display for entering source code or data. The format of the LANGUAGE parameter is:

```
⎡LANGUAGE=⎧FREEFORM⎫⎤
⎢         ⎨FORTRAN ⎬⎥
⎢         ⎪COBOL   ⎪⎥
⎣         ⎩RPG     ⎭⎦
```

LANGUAGE=FREEFORM displays a freeform screen that you can design to accept uniform data or assembler source code. This screen has no fields or tabs set up. It accepts up to 128-character records and truncates them or folds them, depending on your specification for the SCRDSPLY parameter.

LANGUAGE=FORTRAN displays formatted screens designed to help you enter FORTRAN IV source code. The maximum record size for FORTRAN IV source entry records on this screen is 80 characters. EDT performs no syntax checking for FORTRAN IV source code entered on this screen.

If your specification for RECENTRY is SINGLE, then the FORTRAN IV formatted screen contains preset fields for the LABEL (columns 1 to 5), CONTINUATION (column 6), STATEMENT (columns 7 to 72), and IDENTIFICATION (columns 73 to 80) sections of your program.

If your specification for RECENTRY is MULT, then the FORTRAN IV formatted screen has preset tabs at column 1 for the start of the LABEL field, column 6 for the CONTINUATION field, column 7 for the start of the STATEMENT field, and column 73 for the start of the IDENTIFICATION field.

LANGUAGE=COBOL displays formatted screens designed to help you enter COBOL source code. The maximum record size for COBOL source entry records on this screen is 80 characters. EDT performs no syntax checking for COBOL source code entered on this screen.

If your specification for RECENTRY is SINGLE, the COBOL formatted screen contains preset fields for the SEQUENCE (columns 1 to 6), CONTINUATION (column 7), TEXT A (columns 8 to 11), TEXT B (columns 12 to 72), and IDENTIFICATION (columns 73 to 80) sections of your program. These fields take up two lines on the screen, but EDT processes them in the proper columns.

If your specification for RECENTRY is MULT, the COBOL formatted screen has preset tabs at column 1 for the start of the SEQUENCE field, column 7 for the CONTINUATION field, column 8 for the start of the TEXT A field, column 12 for the start of the TEXT B field, and column 73 for the start of the IDENTIFICATION field.

Note that the COBOL screens identify the IDENTIFICATION field for your program, not the IDENTIFICATION DIVISION of your program.

LANGUAGE=RPG displays formatted screens designed to help your enter RPG II source code. The maximum record size for RPG II source entry records on this screen is 80 characters. EDT performs no syntax checking for RPG II source code entered on this screen.

If your specification for RECENTRY is SINGLE, then the RPG II formatted screen contains preset fields for the PAGE-LINE (columns 1 to 5), FORM (column 6), CONTROL-FIELDS (columns 7 to 74), and PROGRAM-ID (columns 75 to 80) sections of your program. These fields take up two lines on the screen, but EDT processes them in their proper columns.

If your specification for RECENTRY is MULT, the RPG II formatted screen has preset tabs at column 1 for the start of the PAGE-LINE field, column 6 for the FORM field, column 7 for the start of the CONTROL-FIELDS, and column 75 for the start of the PROGRAM-ID field.

Each formatted screen contains an EDT command entry line where you can enter any EDT command along with your source code or data. If you enter both, EDT places the source code or data in its work space first, then processes the command.

See 10.2.6 for examples of the formatted screens EDT displays in screen mode.

10.2.5. Defining Your Character Input Prompt (SCRFORM)

The screen format parameter (SCRFORM) tells EDT which format to use as a character input prompt on the screens it displays - either underlines or blanks. The format of the SCRFORM parameter is:

```
[SCRFORM={UNDERLINE}]
[        {BLANK    }]
```

SCRFORM=UNDERLINE tells EDT that you want your entry screens to have underlines as character input prompts. When you transmit these screens, EDT changes your unused underline characters to blanks; therefore, you don't have to be concerned with them. Remember, if you specify SCRFORM=UNDERLINE, you can't use an underline character in any of your source code or data. Instead, you must substitute a different character and later change it to an underline using the EDT CHANGE command.

SCRFORM=BLANK tells EDT that you want your entry screens to have no character input prompts at all. Since you won't be able to see where field settings start, you can simply tab to the next field. When SCRFORM=BLANK, you can enter any character on your EDT commands.

No matter what you specify for SCRFORM, the EDT command entry line on all the screen mode screens contains no character input prompt. We designed screen mode in this way so you can enter the underline as a valid character in any EDT command at any time.

10.2.6. Sample Screen Mode Screens

As you can see from reading the previous five sections, EDT can display a variety of screens to help you enter your source code or data in screen mode. The screens it displays, then, depend on your specifications for the parameters described in 10.2.1 through 10.2.5.

This subsection talks about the types of screens you'll see and use as a result of combining the different values on the parameters and gives some examples. For a discussion of how you actually use screen-mode screens for entering source code or data, see 10.3.

Remember that there are certain areas common to all screen-mode screens. We describe these common areas at the beginning of 10.2.

10.2.6.1. Sample Screens for Entering Data or Assembler Source Code (LANGUAGE=FREEFORM)

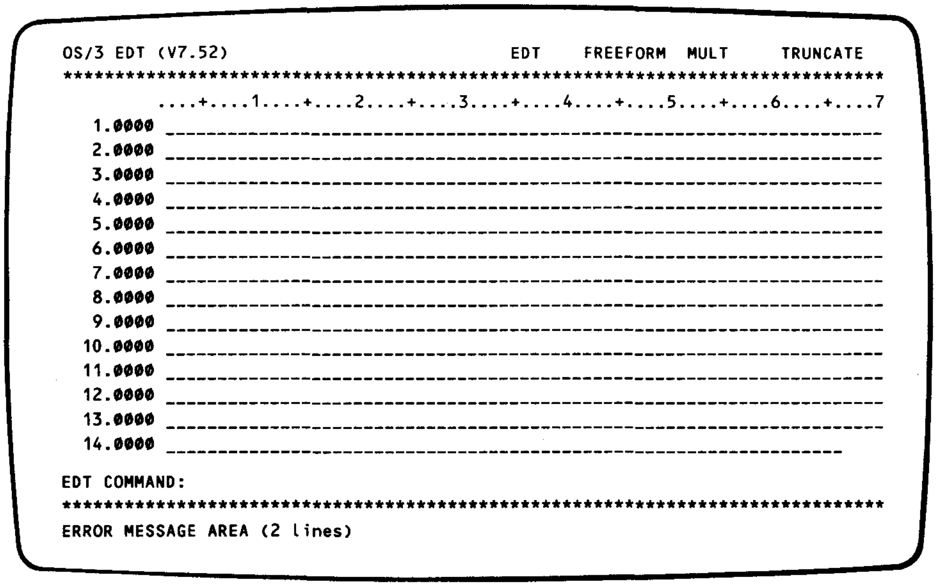
Here, we'll show two typical examples of screens EDT may display to help you enter data or ASSEMBLER source code. Whether EDT displays these types of screens depends on your specifications or default values for the SCRDSPLY, RECENTRY, and SCRFORM parameters of the @SET directive. Because you'll probably want to enter more than one data or ASSEMBLER source line at a time, we give examples where the value for RECENTRY equals MULT, or where the screens accept 1 to 14 lines in one transmission.

Example 1:

Let's say you specify:

```
    @SET MODE=SCREEN
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                          EDT    FREEFORM  MULT     TRUNCATE
******************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
   1.0000 ____________________________________________________________________
   2.0000 ____________________________________________________________________
   3.0000 ____________________________________________________________________
   4.0000 ____________________________________________________________________
   5.0000 ____________________________________________________________________
   6.0000 ____________________________________________________________________
   7.0000 ____________________________________________________________________
   8.0000 ____________________________________________________________________
   9.0000 ____________________________________________________________________
  10.0000 ____________________________________________________________________
  11.0000 ____________________________________________________________________
  12.0000 ____________________________________________________________________
  13.0000 ____________________________________________________________________
  14.0000 ________________________________________________________________
EDT COMMAND:
******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

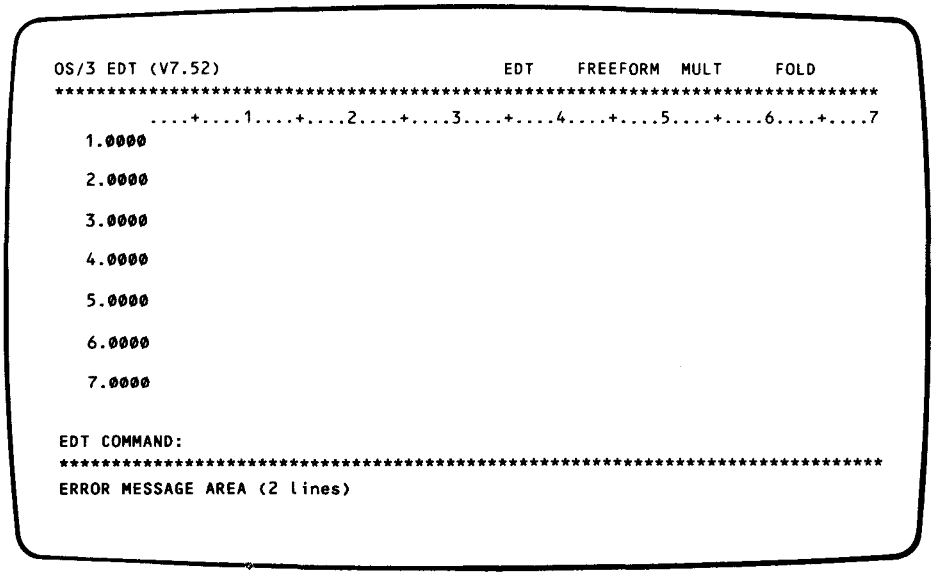
Because you didn't specify the SCRDSPLY, RECENTRY, LANGUAGE, or SCRFORM parameters, EDT uses the default values for those parameters. Therefore, it displays a freeform screen that accepts up to 14 partial lines of data or ASSEMBLER source code and includes underlines as data prompts.

Example 2:

Let's say you specify:

```
@SET SCRDSPLY=FOLD,MODE=SCREEN,SCRFORM=BLANK
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                          EDT    FREEFORM  MULT     FOLD
******************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
   1.0000

   2.0000

   3.0000

   4.0000

   5.0000

   6.0000

   7.0000

EDT COMMAND:
******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Because you didn't specify the RECENTRY or LANGUAGE parameters, EDT uses the default values for those parameters. Therefore, it displays a freeform screen that accepts up to 7 full-character lines of data or ASSEMBLER source code and includes no data prompts.

10.2.6.2. Sample Screens for Entering FORTRAN IV Source Code (LANGUAGE=FORTRAN)

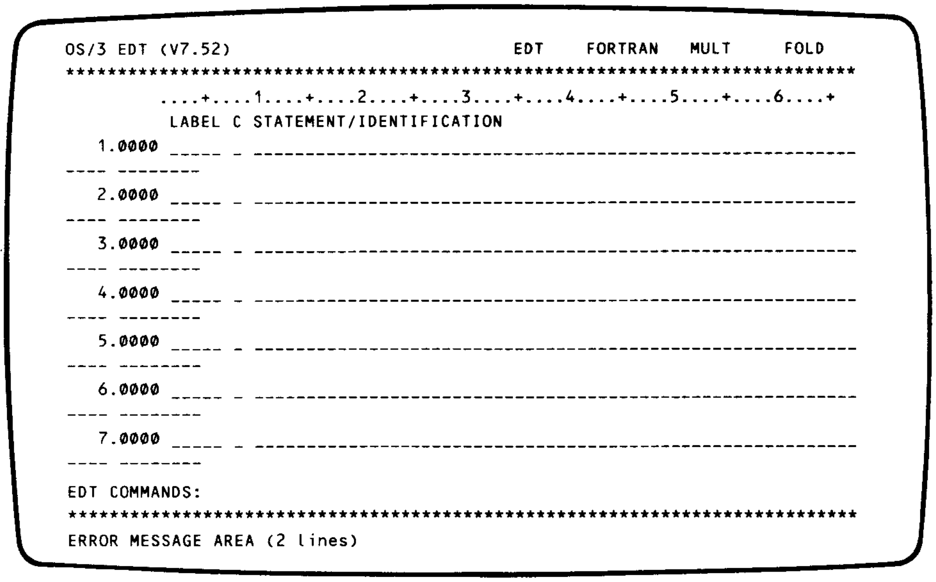
Here, we'll show two typical examples of screens EDT may display to help you enter FORTRAN IV source code. Whether EDT displays these types of screens depends on your specifications or default values for the SCRDSPLY, RECENTRY, and SCRFORM parameters of the @SET directive. Because you'll probably want to enter more than one FORTRAN IV source line at a time, we give examples where the value for RECENTRY equals MULT, or where the screens accept 1 to 14 lines in one transmission.

Example 1:

Let's say you specify:

```
@SET SCRDSPLY=FOLD,MODE=SCREEN,LANGUAGE=FORTRAN
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                            EDT    FORTRAN   MULT     FOLD
*****************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+
          LABEL C STATEMENT/IDENTIFICATION
   1.0000 _____ _ -----------------------------------------------------------
---- --------
   2.0000 _____ _ -----------------------------------------------------------
---- --------
   3.0000 _____ _ -----------------------------------------------------------
---- --------
   4.0000 _____ _ -----------------------------------------------------------
---- --------
   5.0000 _____ _ -----------------------------------------------------------
---- --------
   6.0000 _____ _ -----------------------------------------------------------
---- --------
   7.0000 _____ _ -----------------------------------------------------------
---- --------
EDT COMMANDS:
*****************************************************************************
ERROR MESSAGE AREA (2 lines)
```

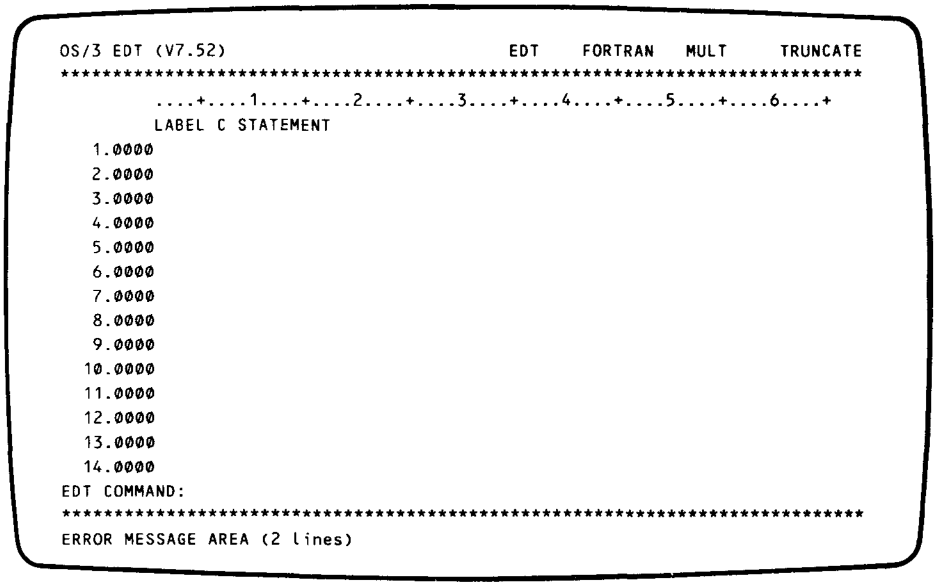
Because you didn't specify the RECENTRY or SCRFORM parameters, EDT uses the default values for those parameters. Therefore, it displays a screen that accepts up to 7 full-character lines of FORTRAN IV source code and includes underlines as data prompts.

Example 2:

Let's say you specify:

```
@SET MODE=SCREEN,LANGUAGE=FORTRAN,SCRFORM=BLANK
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                        EDT    FORTRAN   MULT     TRUNCATE
*****************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+
         LABEL C STATEMENT
  1.0000
  2.0000
  3.0000
  4.0000
  5.0000
  6.0000
  7.0000
  8.0000
  9.0000
 10.0000
 11.0000
 12.0000
 13.0000
 14.0000
EDT COMMAND:
*****************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Because you didn't specify the SCRDSPLY or RECENTRY parameters, EDT uses the default values for those parameters. Therefore, it displays a screen that accepts up to 14 partial lines of FORTRAN IV source code and includes no data prompts.

10.2.6.3. Sample Screens for Entering COBOL Source Code (LANGUAGE=COBOL)

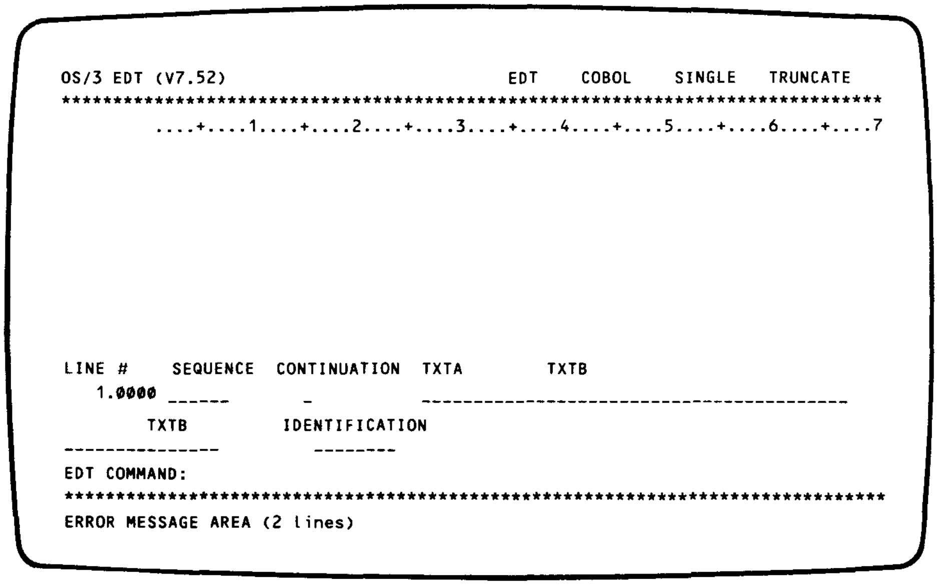
Here we'll show two typical examples of screens EDT can display to help you enter COBOL source code. Whether EDT displays these types of screens depends on your specifications or default values for the SCRDSPLY, RECENTRY, and SCRFORM parameters of the @SET directive. Because you'll probably use screen mode for only updating or correcting (not creating) COBOL source modules, we give examples where the value for RECENTRY equals SINGLE, or where the screens accept only one source line at a time.

Example 1:

Let's say you specify:

```
@SET MODE=SCREEN,RECENTRY=SINGLE,LANGUAGE=COBOL
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                               EDT    COBOL    SINGLE   TRUNCATE
*******************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7







LINE #    SEQUENCE  CONTINUATION  TXTA        TXTB
   1.0000 ______        _         _________________________________________
        TXTB             IDENTIFICATION
_______________     ________
EDT COMMAND:
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

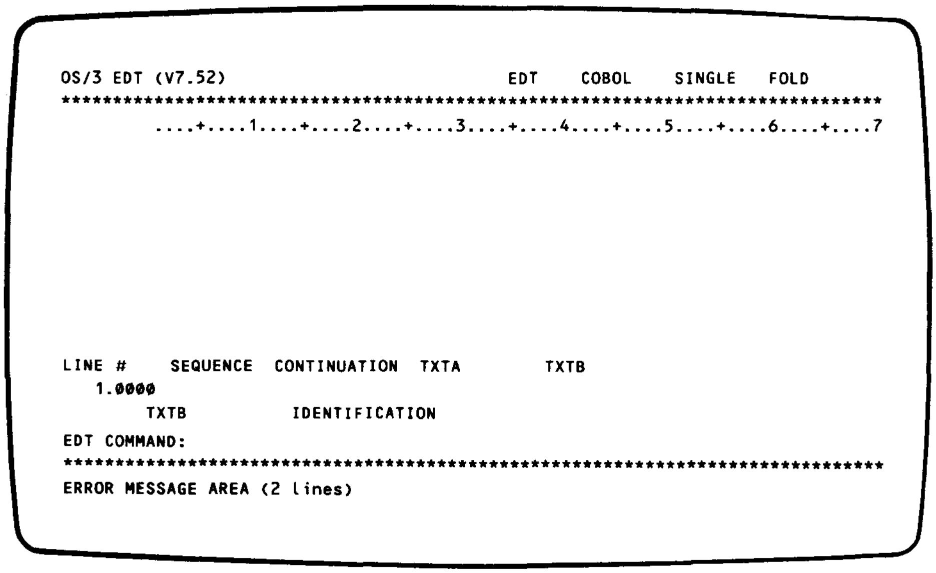
Because you didn't specify the SCRDSPLY or SCRFORM parameters, EDT uses the default values for those parameters. Therefore, it displays a screen that accepts one line of COBOL source code at a time and includes underlines as data prompts. Notice that this screen's display area has room for only a 70-column display and its source entry area accepts 80 characters.

Example 2:

Let's say you specify:

```
@SET SCRDSPLY=FOLD,MODE=SCREEN,RECENTRY=SINGLE,LANGUAGE=COBOL,SCRFORM=BLANK
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                               EDT    COBOL    SINGLE   FOLD
*******************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7







LINE #    SEQUENCE  CONTINUATION  TXTA        TXTB
   1.0000
        TXTB              IDENTIFICATION
EDT COMMAND:
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Because you didn't accept the default values for any of the five parameters that determine your screens, EDT uses your specifications for all the parameters. Therefore, it displays a screen that accepts one full-character line of COBOL source code at a time and includes no data prompts. Notice that this screen's display area has room for a 128-column display, regardless of the fact that COBOL screens accept only 80-character records.

10.2.6.4. Sample Screens for Entering RPG II Source Code (LANGUAGE=RPG)

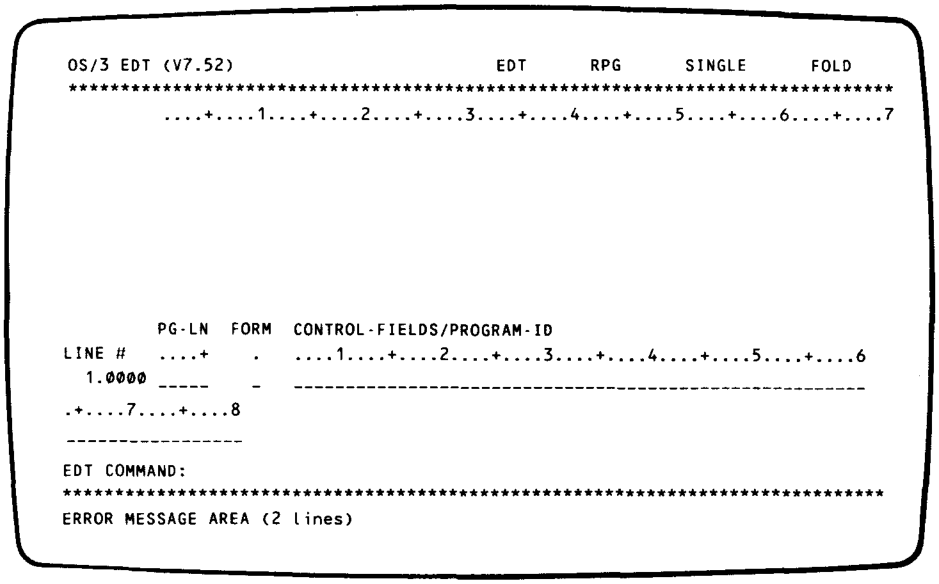
Here we'll show two typical examples of screens EDT displays to help you enter RPG II source code. Whether EDT displays these types of screens depends on your specifications or default values for the SCRDSPLY, RECENTRY, and SCRFORM parameters of the @SET directive. Because you'll probably use screen mode for only updating or correcting (not creating) RPG II source modules, we give examples where the value for RECENTRY equals SINGLE (the screens accept only one source line at a time).

Example 1:

Let's say you specify:

```
@SET SCRDSPLY=FOLD,MODE=SCREEN,RECENTRY=SINGLE,LANGUAGE=RPG
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                         EDT      RPG       SINGLE    FOLD
********************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7






         PG-LN  FORM  CONTROL-FIELDS/PROGRAM-ID
LINE #   ....+    .   ....1....+....2....+....3....+....4....+....5....+....6
  1.0000 _____    _   _______________________________________________________
.+....7....+....8
_________________
EDT COMMAND:
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

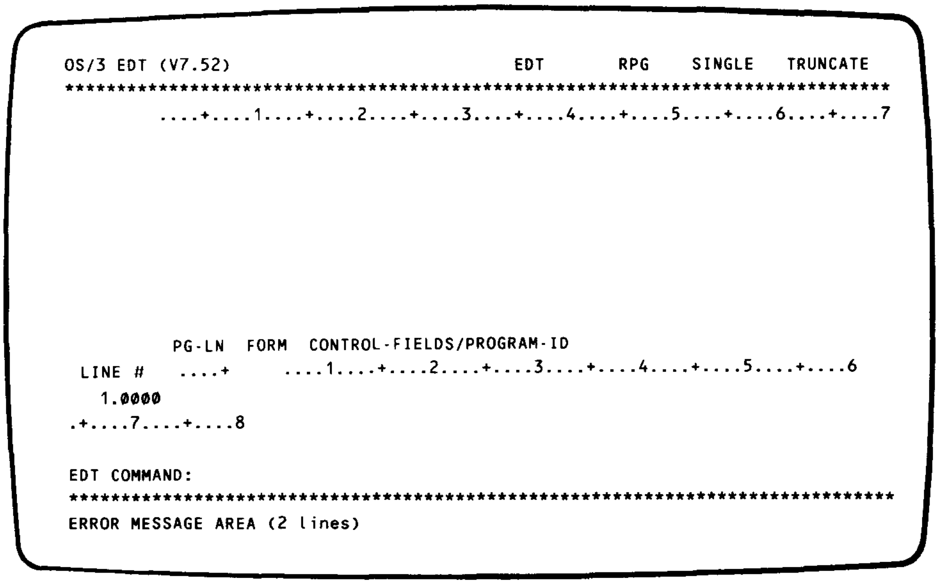
Because you didn't specify the SCRFORM parameter, EDT uses the default value for that parameter. Therefore, it displays a screen that accepts one full-character line of RPG II source code at a time and includes underlines as data prompts. Notice that this screen's display area has room for 128-character display, regardless of the fact that RPG II screens accept only 80-character records.

Example 2:

Let's say you specify:

```
@SET MODE=SCREEN,RECENTRY=SINGLE,LANGUAGE=RPG,SCRFORM=BLANK
```

As soon as you press the XMIT key, EDT displays a screen similar to this one:



```
OS/3 EDT (V7.52)                            EDT       RPG    SINGLE   TRUNCATE
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7









          PG-LN  FORM  CONTROL-FIELDS/PROGRAM-ID
 LINE #   ....+     ....1....+....2....+....3....+....4....+....5....+....6
   1.0000
.+....7....+....8

EDT COMMAND:
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Because you didn't specify the SCRDSPLY parameter, EDT uses the default value for that parameter. Therefore, it displays a screen that accepts one line of RPG II source code at a time and includes no data prompts. Notice that this screen's display area has room for only a 70-column display.

10.3. USING SCREEN MODE

10.3.1. Beginning Your Screen Mode Session

To begin your screen mode session, key in:

```
@SET MODE=SCREEN [parameters]
```

where:

`parameters`

Are any of the four parameters that define the type of screens EDT displays in screen mode. We describe these parameters in detail in 10.2 and give recommendations for specifying them, depending on what you're entering, in 10.3.2.1 through 10.3.2.4.

Once you press the XMIT key after entering the MODE parameter of the @SET directive, EDT immediately displays a screen of the type you've defined and you're in screen mode. From this point on, you can do any of the following:

- enter source code or data (see 10.3.2);
- enter EDT commands (10.3.3);
- display EDT's work-space contents (10.3.4); or
- end your screen mode session (10.3.5).

10.3.2. Entering Source Code or Data

As soon as you enter the @SET MODE=SCREEN directive, EDT displays a screen-mode screen of the type you've defined.

From this screen, you enter source code or data. Each time you press the XMIT key, EDT displays another screen of the same type. EDT keeps displaying the same type of screen until you do one of the following:

- redefine your type of screens;
- end your screen mode session; or
- terminate EDT.

Naturally, the way you use the screen mode screens differs, depending on whether you're entering source code or data and on how you defined your screens. And if you're entering source code, the method depends on whether you're entering FORTRAN IV, COBOL, or RPG II source code. The following subsections, therefore, discuss entering data and each type of source code using screen-mode screens.

However, there are certain considerations you should be aware of no matter what you're entering in screen mode. They are:

- When EDT displays a new screen-mode screen, the cursor is always positioned at the next available work-space line number. You can change this line number using the EDT @ command, but EDT interprets the line number at the time of transmission as the current line number.

- You can't enter source code or data from the EDT command entry line on any screen-mode screens, nor can you use the double at sign (@@) before a data line to enter data from the EDT command entry line. EDT flags as invalid anything entered from this line except an EDT command.

- Before pressing the XMIT key on any screen, be sure that you move the cursor to the bottom rightmost position on the screen to ensure that EDT transmits all your source code or data to its work-space file.

- From any screen you can enter both source code or data and an EDT command. If entering both, EDT places the source code or data in its work space first, then processes the EDT command. If you enter neither, and your screens accept single record entries, EDT places a blank line in its work space after you press the XMIT key.

- If your screens contain underlines as entry prompts, don't include underlines in your source code or data entries. Instead, use another character. Then, go back and change that character to an underline using the EDT CHANGE command. If you accidentally include an underline, EDT processes it as though it were a blank character.

- If your screens don't contain underlines as entry prompts, tab to the various programming fields on the screens since you can't see where the fields begin and end.

- Regardless of whether your screens use underlines as entry prompts, the EDT command entry line never includes underlines. We designed screen mode in this way so you can enter the underline character in any EDT command at any time.

- If your screens accept single-record truncated entries, their display areas show the last 10 lines of source code or data you entered. If they accept single full-character records, your screens' display areas show the last five lines of source code or data, because full-character records take up twice as many lines as truncated records.

- If your screens accept multiple records, EDT places embedded blank lines in its work space as blank lines; however, it doesn't place trailing blank lines in its work space. For example, if on one screen you enter four lines of data, skip line 5, enter two more lines of data, and press the XMIT key, EDT places lines 1 through 7 in its work space. The next screen EDT displays will show the current line number as 8.0000.

- If you're using screen mode to update a module or file, you must enter a READ command to copy the module or file you want to update from its permanent file to the EDT work space. Next, enter a PRINT command to display the module or file on your workstation screen and note the lines that need to be updated. You can enter these commands before or after you issue the SET directive to enter screen mode.

- If you enter more characters than a screen mode field accepts, EDT automatically tabs to the next field. In these cases, use the TAB BACK key to go back to the field and enter the correct number of characters.

10.3.2.1. Entering Data or Assembler Source Code

To enter data or Assembler source code, you should have begun your screen mode session (10.3.1) with the value for the LANGUAGE parameter of the @SET directive equal to FREEFORM (LANGUAGE=FREEFORM). Because FREEFORM is the default for that parameter, you didn't have to specify it explicitly when you began your screen mode session.

When keying in the @SET directive to begin your screen mode session, you should also specify the various parameters for the directive in the following way:

- If your lines of input are longer than 70 characters across, specify SCRDSPLY as FOLD.

- If you're creating a data file or Assembler module, be sure the value for the RECENTRY parameter is MULT for multiple record entry. If you're merely updating an existing data file or Assembler module, you may want to specify RECENTRY as SINGLE to enter one line of data or assembler code in each transmission.

- If your lines of input contain underlines, specify SCRFORM as BLANK.

- If you want your data in columns, use the CHAR and TABS parameters to set your tab stops and identify your tab key.

See 10.2 for a complete description of the SCRDSPLY, RECENTRY, and SCRFORM parameters and 7.9 for a complete description of the TABS and CHAR parameters.

As soon as you transmit the @SET directive to begin your screen mode session, EDT displays a freeform screen whose characteristics depend on your specifications for the SCRDSPLY, RECENTRY, and SCRFORM parameters. The cursor is positioned at the first input field on your workstation screen.

When EDT displays this freeform screen, enter your lines of data or Assembler code. Although you can't enter column headings for your input, you can have EDT process them in column format. You do this by pressing your tab key (specified by the CHAR parameter) each time you want to enter an item at a preset tab stop (specified by the TABS parameter).

When using the freeform screens, the EDT tab function operates as we describe in 7.9. That is, when you actually press your tab character while entering input, EDT doesn't move the cursor to the appropriate tab the way it does on the formatted screens for FORTRAN IV, COBOL, and RPG II (or the way a typewriter does). However, in the future, whenever EDT displays your input on the workstation screen or prints a listing of it, the input will be in the correct tab format you designed.

The way you enter your input lines depends on your value for the RECENTRY parameter:

- If your value for RECENTRY is SINGLE, press the XMIT key after each line you enter.
- If your value for RECENTRY is MULT, press the TAB FORWARD key after each line to position the cursor at the beginning of the next line. Then, when you've entered your entire screen (no more than 14 lines), press the XMIT key.

Either method results in EDT displaying another screen of the same type. Keep following one of these methods until you've entered all the lines of data or Assembler code you have. When you're finished, you can perform any of the functions we describe in 10.3.1. See 10.4 for a sample screen mode session for creating a data file and Appendix G for a sample session for updating a MIRAM data file in screen mode.

10.3.2.2. Entering FORTRAN IV Source Code

To enter FORTRAN IV source code, you should have begun your screen mode session (10.3.1) with the value for the LANGUAGE parameter of the @SET directive equal to FORTRAN (LANGUAGE=FORTRAN). When keying in the directive to begin your session, you should also specify the various parameters in the following way:

- If your FORTRAN IV source lines are longer than 70 characters, specify SCRDSPLY as FOLD.
- If you're creating a FORTRAN IV source module, be sure the value for the RECENTRY parameter is MULT for multiple record entry. If you're merely updating an existing FORTRAN IV source module, you may want to specify RECENTRY as SINGLE to enter one line of FORTRAN IV source code in each transmission.
- If your FORTRAN IV source code contains underlines, specify SCRFORM as BLANK.

See 10.2 for a complete description of the SCRDSPLY, RECENTRY, and SCRFORM parameters.

As soon as you transmit the @SET directive to begin your screen mode session, EDT displays a FORTRAN IV screen whose characteristics depend on your specifications for the SCRDSPLY, RECENTRY, and SCRFORM parameters. The cursor is positioned at the first input field on your workstation screen.

When EDT displays your FORTRAN IV screen, you enter your source code in one of the following ways, depending on your value for the RECENTRY parameter:

- If the value for RECENTRY is SINGLE, the FORTRAN IV screen contains preset and marked fields for the LABEL (columns 1-5), CONTINUATION (column 6), STATEMENT (columns 7-72), and IDENTIFICATION (columns 73-80) sections of your program. Enter each line of your source code, pressing the TAB FORWARD key to move to the next field. At the end of each line, press the XMIT key. EDT then displays another FORTRAN IV screen of the same type.

- If the value for RECENTRY is MULT, the FORTRAN IV screen has preset tabs at column 1 for the start of the LABEL field, column 6 for the CONTINUATION field, column 7 for the start of the STATEMENT field, and column 73 for the start of the IDENTIFICATION field. The screen accepts up to 14 truncated lines of FORTRAN IV source code or 7 full-character lines.

  Enter each line of your source code, pressing the TAB FORWARD key to move the next field. At the end of each line, press the TAB FORWARD key to move to the next line. When you've entered your entire screen (up to 14 lines), press the XMIT key. EDT then displays another FORTRAN IV screen of the same type.

Keep following one of these methods until you've entered all your FORTRAN IV code. When you're finished, you can perform any of the functions described in 10.3.1.

10.3.2.3. Entering COBOL Source Code

To enter COBOL source code, you should have begun your screen mode session (10.3.1) with the value for the LANGUAGE parameter of the @SET directive equal to COBOL (LANGUAGE=COBOL). When keying in the directive to begin your session, you should also specify the various parameters in the following way:

- If your COBOL source code lines are longer than 70 characters across, specify SCRDSPLY as FOLD.

- Because you'll probably use screen mode for only updating or correcting (not creating) COBOL source modules, specify RECENTRY as SINGLE. Of course, if you want to use screen mode instead of the COBOL editor to create COBOL source modules, specify RECENTRY as MULT for multiple record entry.

- If your COBOL source code contains underlines, specify SCRFORM as BLANK.

See 10.2 for a complete description of the SCRDSPLY, RECENTRY and SCRFORM parameters.

As soon as you transmit the @SET directive to begin your screen mode session, EDT displays a COBOL screen whose characteristics depend on your specifications for the SCRDSPLY, RECENTRY, and SCRFORM parameters. The cursor is positioned at the first input field on your workstation screen.

When EDT displays your COBOL screen, you enter your source code in one of the following ways, depending on your value for the RECENTRY parameter:

- If the value for RECENTRY is SINGLE, the COBOL screen contains preset and marked fields for the SEQUENCE (columns 1–6), CONTINUATION (column 7), TEXT A (columns 8–11), TEXT B (columns 12–72), and IDENTIFICATION (columns 73–80) sections of your program. These fields take up two lines of your screen but EDT processes them in their proper columns. Enter each line of your source code, pressing the TAB FORWARD key to move to the next field. At the end of each line, press the XMIT key. EDT then displays another COBOL screen of the same type. Note that IDENTIFICATION refers to the identification of your program, not its IDENTIFICATION DIVISION.

- If the value for RECENTRY is MULT, the COBOL screen has preset tabs at column 1 for the start of the SEQUENCE field, column 7 for the CONTINUATION field, column 8 for the start of the TEXT A field, column 12 for the start of the TEXT B field, and column 73 for the start of the IDENTIFICATION field. Note that IDENTIFICATION refers to the identification of your program, not its IDENTIFICATION DIVISION. The COBOL screen accepts up to 14 truncated lines of COBOL source code or 7 full-character lines.

  Enter each line of your code, pressing the TAB FORWARD key to move to the next field. At the end of each line, press the TAB FORWARD key to move to the next line. When you've entered your entire screen (up to 14 lines), press the XMIT key. EDT then displays another COBOL screen of the same type.

Keep following one of these methods until you've entered all your COBOL code. When you're finished, you can perform any of the functions described in 10.3.1. See Appendix G for a sample screen mode session for updating a COBOL source module.

10.3.2.4. Entering RPG II Source Code

To enter RPG II source code, you should have begun your screen mode session (10.3.1) with the value for the LANGUAGE parameter of the @SET directive equal to RPG II (LANGUAGE=RPG II). When keying in the directive to begin your session, you should also specify the various parameters in the following way:

- If your RPG II source code lines are longer than 70 characters across, specify SCRDSPLY as FOLD.

- Because you'll probably use screen mode for only updating or correcting (not creating) RPG II source modules, specify RECENTRY as SINGLE. Of course, if you want to use screen mode instead of the RPG II editor to create an RPG II source module, specify RECENTRY as MULT for multiple record entry.

- If your RPG II source code contains underlines, specify SCRFORM as BLANK.

See 10.2 for a complete description of the SCRDSPLY, RECENTRY, and SCRFORM parameters.

As soon as you transmit the @SET directive to begin your screen mode session, EDT displays an RPG II screen whose characteristics depend on your specifications for the SCRDSPLY, RECENTRY, and SCRFORM parameters. The cursor is positioned at the first input field on your workstation screen.

When EDT displays your RPG II screen, you enter your source code in one of the following ways, depending on your value for the RECENTRY parameter:

- If the value for RECENTRY is SINGLE, the RPG II screen contains preset and marked fields for the PAGE-LINE (columns 1–5), FORM (column 6), CONTROL-FIELDS (columns 7–74), and PROGRAM-ID (columns 75–80) sections of your program. These fields take up two lines on your screen, but EDT processes them in their proper columns. Enter each line of your source code, pressing the TAB FORWARD key to move to the next field. At the end of each line, press the XMIT key. EDT then displays another RPG II screen of the same type.

- If the value for RECENTRY is MULT, the RPG II screen has preset tabs at column 1 for the start of the PAGE-LINE field, column 6 for the FORM field, column 7 for the start of the CONTROL-FIELDS, and column 75 for the start of the PROGRAM-ID field. The screen accepts up to 14 truncated lines of RPG II source code or 7 full-character lines.

  Enter each line of your code, pressing the TAB FORWARD key to move to the next field. At the end of each line, press the TAB FORWARD key to move to the next line. When you've entered your entire screen (up to 14 lines), press the XMIT key. EDT then displays another RPG II screen of the same type.

Keep following one of these methods until you've entered all your RPG II code. When you're finished, you can perform any of the functions described in 10.3.1.

10.3.3. Entering EDT Commands in Screen Mode

From any screen mode screen, you can enter any EDT command (except the UPDATE command). You enter EDT commands (meaning commands, modifiers, procedure file commands, variables, directives, screen commands, and function keys) on the EDT command entry line, which appears on every screen-mode screen.

On the EDT command entry line, you can enter only EDT commands. EDT flags as invalid anything entered from this line except an EDT command.

Because the command entry line never includes underlines as prompts, you can always use the underline in your EDT commands.

From any screen, you can enter both source code or data and an EDT command. If entering both, EDT places the source code or data in its work space first, then processes the EDT command. If you enter neither one and your screens accept single record entries, EDT places a blank line in its work space after you press the XMIT key.

If you want to enter the PRINT command, the CHECK directive, or the ROLL screen command specifically to display the contents of EDT's work space, see 10.3.4.

10.3.4. Displaying EDT's Work-Space Contents in Screen Mode

At any point in your screen mode session, you can display, on your workstation screen, the contents of the EDT work space to view the source code or data you've already entered. Just as in line mode, you display the work-space contents by entering an @PRINT command, an @CHECK directive, or an @ROLL screen command. In screen mode, you enter any of these commands on the EDT command entry line that is on all screen-mode screens.

The @PRINT command displays specified lines from the work-space file on the workstation screen. The @CHECK directive determines whether EDT should display lines that it processed as a result of another EDT command. The @ROLL screen command lets you roll through EDT's work-space file, displaying lines, and if you want, updating or correcting them.

In all cases, EDT displays the work-space contents according to your specification for the SCRDSPLY parameter of the @SET directive. If SCRDSPLY=TRUNCATE, EDT displays one record per one screen line and truncates lines if necessary; if SCRDSPLY=FOLD, EDT displays one record per two screen lines and doesn't truncate lines.

After EDT prints a full screen or the last screen, you have four options:

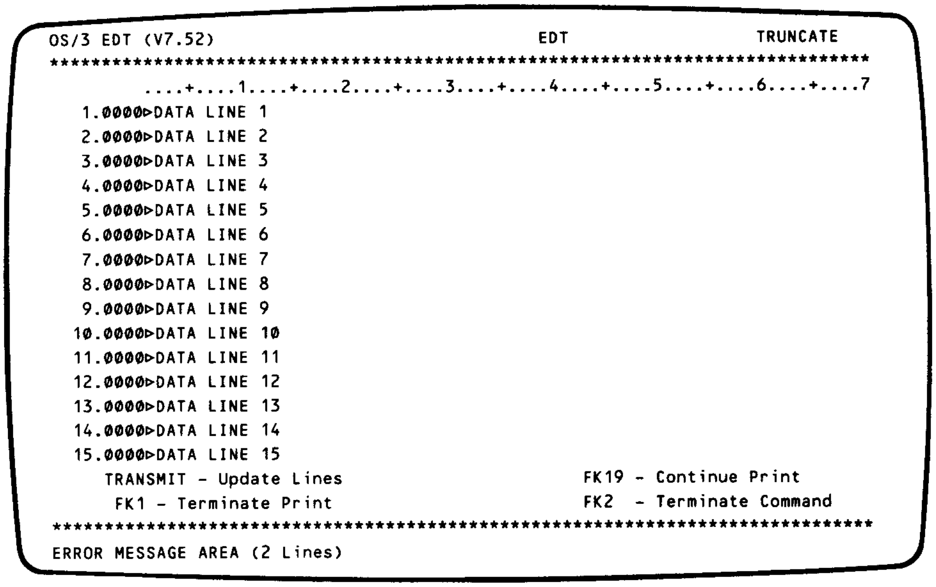
1. You may update the displayed lines and transmit them to the EDT work space.

2. You may continue the PRINT command, CHECK directive, or ROLL screen command (i.e., display remaining specified lines) by pressing the F19 function key.

3. You may prevent the display of succeeding lines by pressing either F1 or F18, then pressing the F19 function key.

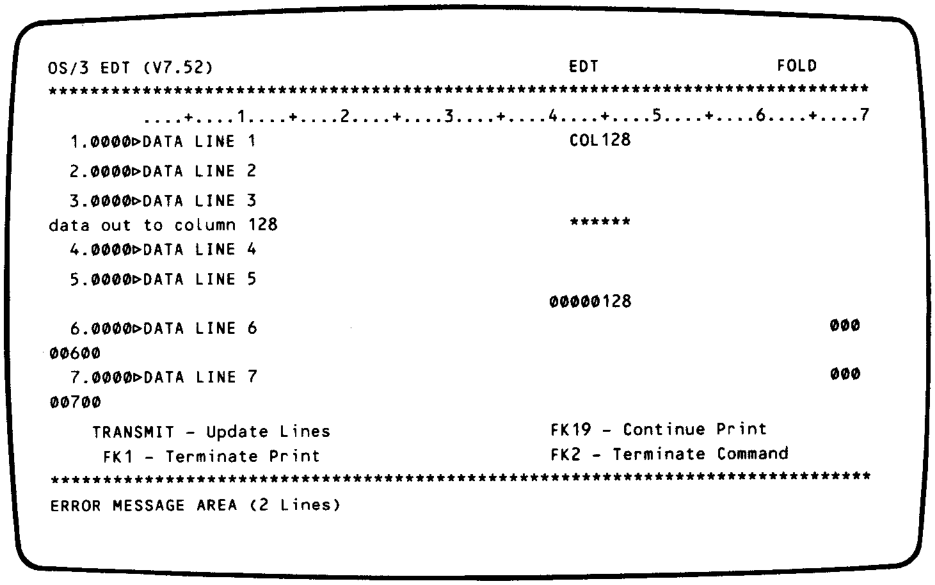
4. You may terminate the entire PRINT command, CHECK directive, or ROLL screen command by pressing function key F2.

The following is an example of an EDT display if SCRDSPLY=TRUNCATE and you issued either an @PRINT command, an @CHECK directive, or an @ROLL screen command:



```
OS/3 EDT (V7.52)                                EDT                  TRUNCATE
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
   1.0000▷DATA LINE 1
   2.0000▷DATA LINE 2
   3.0000▷DATA LINE 3
   4.0000▷DATA LINE 4
   5.0000▷DATA LINE 5
   6.0000▷DATA LINE 6
   7.0000▷DATA LINE 7
   8.0000▷DATA LINE 8
   9.0000▷DATA LINE 9
  10.0000▷DATA LINE 10
  11.0000▷DATA LINE 11
  12.0000▷DATA LINE 12
  13.0000▷DATA LINE 13
  14.0000▷DATA LINE 14
  15.0000▷DATA LINE 15
      TRANSMIT - Update Lines                         FK19 - Continue Print
       FK1 - Terminate Print                          FK2  - Terminate Command
********************************************************************************
ERROR MESSAGE AREA (2 Lines)
```

The following is an example of an EDT display if SCRDSPLY=FOLD and you issued either an @PRINT command, an @CHECK directive, or an @ROLL screen command:



```
OS/3 EDT (V7.52)                                   EDT                    FOLD
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  1.0000▷DATA LINE 1                               COL128
  2.0000▷DATA LINE 2
  3.0000▷DATA LINE 3
data out to column 128                             ******
  4.0000▷DATA LINE 4
  5.0000▷DATA LINE 5
                                                 00000128
  6.0000▷DATA LINE 6                                                          000
00600
  7.0000▷DATA LINE 7                                                          000
00700
    TRANSMIT - Update Lines                      FK19 - Continue Print
     FK1 - Terminate Print                       FK2 - Terminate Command
********************************************************************************
ERROR MESSAGE AREA (2 Lines)
```

10.3.5. Ending Your Screen Mode Session

After you've entered all your source code or data, you can either:

- end your screen mode session but continue your EDT session in line mode; or
- terminate your entire EDT session.

To continue operating in line mode, on the EDT command entry line of your last screen mode screen, key in:

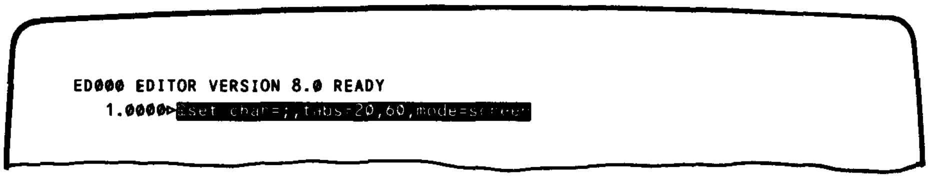
```
@SET MODE=LINE
```

and press the XMIT key. EDT then displays the current line number and positions the cursor to accept another EDT command or a data line.

To terminate your entire EDT session, first enter an @WRITE command to write the source code or data you entered in screen mode from the EDT work space to a permanent file. Then, enter an @HALT directive to terminate your entire EDT session.

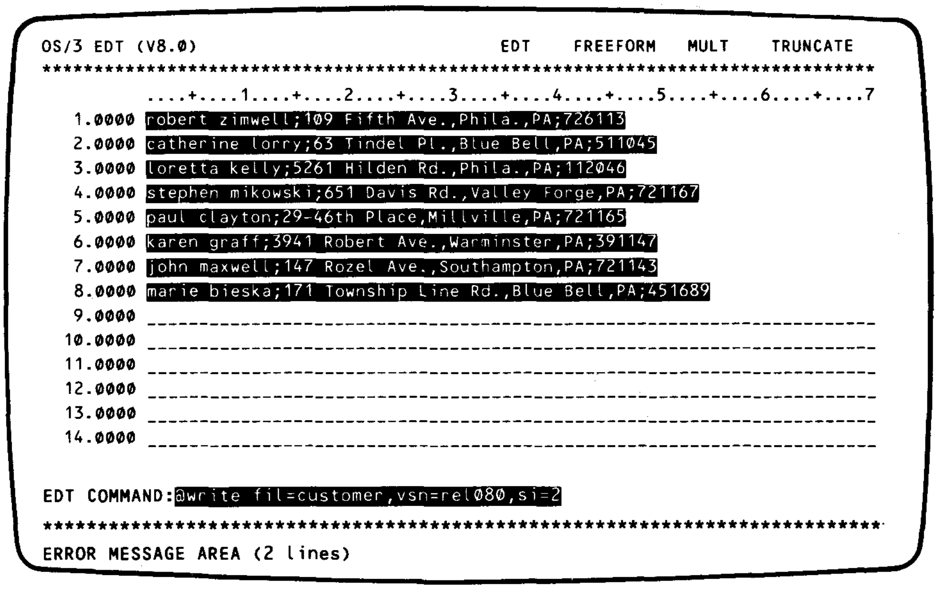
10.4. SAMPLE SCREEN MODE SESSION

In this sample screen mode session, we will create a MIRAM data file using the screen-mode freeform screen. The file will contain a list of customer names, addresses, and account numbers. All your entries are shown in white letters on a black background.



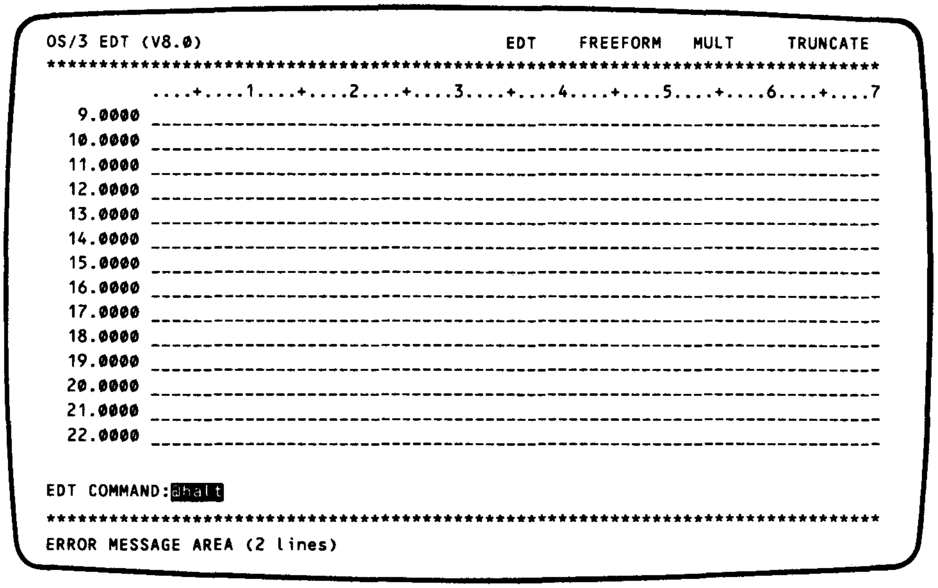
```
ED000 EDITOR VERSION 8.0 READY
   1.0000▷@set char=;,tabs=20,60,mode=screen
```

After you activate EDT (as we describe in Section 2), EDT displays a message telling you that it's ready and displays the first work-space line. You enter an @SET directive to set your tab character to a semicolon and to set your tabs at column 20 and column 60. Setting the tabs lets us format our file into columns containing the names, addresses, and account numbers of our customers. You also include the MODE=SCREEN parameter to enter screen mode. Then you press the XMIT key.



```
OS/3 EDT (V8.Ø)                            EDT    FREEFORM   MULT    TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  1.ØØØØ robert zimwell;109 Fifth Ave.,Phila.,PA;726113
  2.ØØØØ catherine lorry;63 Tindel Pl.,Blue Bell,PA;511Ø45
  3.ØØØØ loretta kelly;5261 Hilden Rd.,Phila.,PA;112Ø46
  4.ØØØØ stephen mikowski;651 Davis Rd.,Valley Forge,PA;721167
  5.ØØØØ paul clayton;29-46th Place,Millville,PA;721165
  6.ØØØØ karen graff;3941 Robert Ave.,Warminster,PA;391147
  7.ØØØØ john maxwell;147 Rozel Ave.,Southampton,PA;721143
  8.ØØØØ marie bieska;171 Township Line Rd.,Blue Bell,PA;451689
  9.ØØØØ ----------------------------------------------------------------------
 1Ø.ØØØØ ----------------------------------------------------------------------
 11.ØØØØ ----------------------------------------------------------------------
 12.ØØØØ ----------------------------------------------------------------------
 13.ØØØØ ----------------------------------------------------------------------
 14.ØØØØ ----------------------------------------------------------------------

EDT COMMAND:@write fil=customer,vsn=relØ8Ø,si=2
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays a screen-mode freeform screen and positions the cursor at the first field on line 1. You enter each name in your data file, press the semicolon (tab) to tell EDT to go to column 20 in its work space, enter each customer's address, press the semicolon key again to go to column 60 in the work space, and enter each customer's account number. After each line, you press the TAB FORWARD key to move the cursor to the next line on the screen. After the eighth data entry, you press the TAB FORWARD key to the EDT COMMAND line, where you enter an @WRITE command to write your MIRAM data file named CUSTOMER to the disk volume whose serial number is rel080. You include the SIZE parameter to allocate the file. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.



```
OS/3 EDT (V8.0)                                   EDT    FREEFORM   MULT     TRUNCATE
********************************************************************************
           ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
   9.0000 ______________________________________________________________________
  10.0000 ______________________________________________________________________
  11.0000 ______________________________________________________________________
  12.0000 ______________________________________________________________________
  13.0000 ______________________________________________________________________
  14.0000 ______________________________________________________________________
  15.0000 ______________________________________________________________________
  16.0000 ______________________________________________________________________
  17.0000 ______________________________________________________________________
  18.0000 ______________________________________________________________________
  19.0000 ______________________________________________________________________
  20.0000 ______________________________________________________________________
  21.0000 ______________________________________________________________________
  22.0000 ______________________________________________________________________

EDT COMMAND:@halt
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another freeform screen and positions the cursor at line 9 since you entered only eight data lines on the previous screen. You press the TAB FORWARD key to the EDT COMMAND line and enter an @HALT directive to end your EDT session. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.

11. Error File Processing

11.1. WHAT IS THE ERROR FILE PROCESSOR (EFP)?

The error file processor, or EFP, is a subroutine of the OS/3 general editor that lets you see errors in your source code from the workstation, provided you had your language compiler create an error file during compilation. EFP gives you the ability to see your source code errors immediately after the language compiler you're using compiles your program, rather than after the compiler prints an error listing.

Because EFP is an interactive product, your source module must reside in a SAT library file for you to use EFP to correct it. Your error file, on the other hand, must have been allocated as a MIRAM file.

As soon as you see your errors, EFP lets you correct them using EDT. Because you're compiling and debugging your program interactively from your workstation, EFP helps improve the turn-around time between compilations.

Because EFP is a subroutine of the general editor, you can also use EFP in a batch environment. See 2.4 for information on using EDT in a batch environment.

11.2. HOW DOES EFP WORK?

EFP is a subroutine of EDT that lets you see and correct source code errors without having the language compiler output listing. Instead of waiting for your compiler to print its listing, you have EFP read your error file. Then, EFP locates the source module to which it applies and shows you both the errors and the source code lines that contain them. At your workstation, using the EDT commands, you can then correct the source code easily and interactively – either one line at a time or several at a time. EFP also lets you simply find out what the errors are without correcting them immediately, in case you don't have time to fix them all. Later, you can run EFP to redisplay the errors and source lines to which they apply and correct them.

NOTE:

If your source program resides on cards, EFP gives a summary of the program's compilation errors; however, you can't correct those errors using EFP. Instead, you must correct the cards and recompile your source module.

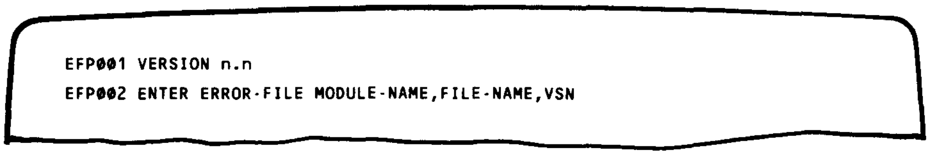
Likewise, if you correct your source module using your compiler's option for correcting the module at the same time you're compiling it, you can't correct its errors using EFP. In this case, you may change the correction deck or use either the SAT librarian (LIBS) or EDT to correct your source module. The system service programs user guide/programmer reference describes the SAT librarian.

11.3. HOW DO I USE EFP?

11.3.1. Activating EFP

You activate EFP using the EDT directive, @EFP. If you're not currently using the editor, you key in EDT @EFP. Before activating EFP, however, empty the EDT work space and make the current line number and increment equal to 1. To empty the EDT work space, use the EDT command, @DELETE. To set the line number and increment to 1 (which they are unless you changed them yourself), use the @ command. These items are necessary so that EFP can synchronize the errors in the error file with the source lines in the source module.

Once you activate EFP, it displays the following messages:



```
EFPØØ1 VERSION n.n
EFPØØ2 ENTER ERROR-FILE MODULE-NAME,FILE-NAME,VSN
```

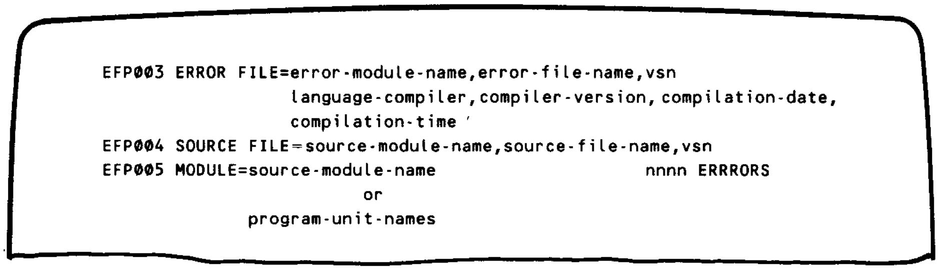
where:

EFPØØ1
: Is message that identifies the EFP version number (n.n), for example, 8.0. This version number may differ from the EDT version number.

EFPØØ2
: Is the message requesting that you identify the MIRAM error file for your program. Be sure to specify the three items EFP requests in the order they appear in the message (module-name, file-name, vsn) and separate them by commas. If the error file name (the LBL name of the file) contains commas, spaces, or parentheses, enclose it in apostrophes or quotes. After you respond to message EFP002, press the XMIT key.

Using the information you supply, EFP then:

- copies the error file to a temporary work file that lasts for the duration of the EFP session;
- identifies the source module for which the error file reports errors;
- scans the error file and reports an error summary at your workstation; and
- displays the following error summary screen:



```
EFPØØ3 ERROR FILE=error-module-name,error-file-name,vsn
                  language-compiler,compiler-version, compilation-date,
                  compilation-time '
EFPØØ4 SOURCE FILE=source-module-name,source-file-name,vsn
EFPØØ5 MODULE=source-module-name                      nnnn ERRRORS
                         or
              program-unit-names
```

where:

EFPØØ3

Is the message that identifies the information you supplied in response to message EFP002, as well as header information that your language compiler supplies about the compilation of your program.

EFPØØ4

Is the message that indicates the source module you are correcting, as well as the file that contains it and the volume that contains the file. EFP automatically identifies the source module using your response to message EFP002; you don't have to specify identification information for the source module.

EFPØØ5

Is the message that identifies the number of errors EFP reports for either:

- the source module if you're correcting a COBOL or RPG II source module; or
- the program unit names if you're correcting a FORTRAN IV source module.

NOTE:

If you're a FORTRAN IV programmer and your compiler processed multiple source modules, EFP displays an EFP004 message for each source module. Following each EFP004 message, EFP displays one EFP005 message for each program unit in each source module.

Using the information in EFP004, EFP directs EDT to read your source module into the EDT work space, which lasts for the duration of the EDT session. Hereafter, you control your source module through regular EDT commands and you control your error file through EFP commands. Once EFP displays the cursor to you, you can enter any EDT command to correct your source module. If you're correcting a COBOL or RPG II source program that needs extensive corrections, you can even use the appropriate subeditor, either COBEDT or RPGEDT, to fix the errors.

The important thing to remember is that EFP lets you enter as many EDT commands as you want to fix your source code before entering another EFP command.

The following subsections describe the EFP commands.

11.3.2. EFP Commands

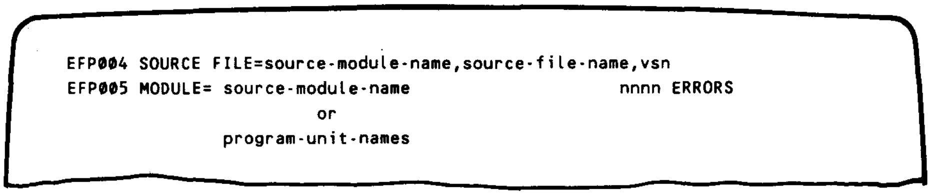
There are three EFP commands you can use to control your error file: EFP SUMMARY, EFP, and EFP END. To see a workstation display of the three commands, use the EDT @PROMPT screen command with the EDT directive EFP; specifically, key in @PROMPT EFP. With this exception only, you can't combine any EFP command with any other EDT commands, nor can you use modifiers with the EFP commands.

11.3.2.1. Requesting an Error File Summary (SUMMARY)

The EFP summary command lets you request an error file summary for the source module. The format of the summary command is:

```
@EFP SUMMARY
 --  ---
```

When you issue this command, EFP displays an error summary screen for the source module you're correcting. The error summary screen looks like this:



```
EFPØØ4 SOURCE FILE=source-module-name,source-file-name,vsn
EFPØØ5 MODULE= source-module-name                      nnnn ERRORS
                        or
                 program-unit-names
```

where:

EFPØØ4
: Is the message that indicates the source module you're correcting, as well as the file that contains it and the volume that contains the file. As we said, EFP automatically identifies the source module using your response to message EFP002; you don't have to specify the identification information for the source module.

EFPØØ5

Is the message that identifies the number of errors EFP reports for either:

- the source module if you're correcting a COBOL or RPG II source module; or
- the program unit names if you're correcting a FORTRAN IV source module.

NOTE:

If you're a FORTRAN IV programmer and your compiler processed multiple source modules, EFP displays an EFP004 message for each source module. Following each EFP004 message, EFP displays one EFP005 message for each program unit in each source module.

After EFP displays the error summary, it positions the cursor to accept another command – either an EFP or an EDT command.

11.3.2.2. Asking EFP to Display Errors (EFP)

The EFP command lets you display the errors in your error file along with the source lines that contain those errors. Note that EFP is both an EDT directive and an EFP command. As an EDT directive, it activates the error file processor; as an EFP command, it displays errors and the lines that contain them so you can correct them.

The EFP command has two formats. Format 1 lets you correct and display COBOL and RPG II errors and FORTRAN IV errors for one source module at a time. By using optional parameters associated with Format 1 of the EFP command, you can specify exactly which errors you want to see and correct. Format 2 concerns only FORTRAN IV programmers who want to see and correct errors for multiple source modules in their compilation. COBOL and RPG II programmers never use Format 2.

Format 1:

```
@EFP[X]Δ[program-unit-name]Δ[error-range]Δ['search-string']
```

where:

X

Specifies that EFP excludes any errors it already displayed in this execution of EFP. When specifying exclusion, don't insert a space between the X and the EFP command.

Without exclusion, @EFP displays the next error in sequence, along with its corresponding source line, even if it already displayed the error in this execution of EFP. For example, even if you've already seen the third error, which applies to line 6 of your source code (which is 100 lines long), the @EFP command tells EFP to display:

```
ERR-ØØ3 error message
   6.ØØØØ source line
 1Ø1.ØØØØ ▷
```

The cursor is positioned on line 101 because your source module contains 100 lines. EFP is ready to accept the next command – either an EDT command to correct the source code or another EFP command. After it displays all the errors in your error file, the EFP command without exclusion tells EFP to start the process over with the first error in the file.

With exclusion, @EFPX displays the next error in sequence, along with its corresponding source line, but only if it hasn't already displayed it in this execution of EFP. For example, if you have already seen the third error in the error file, the @EFPX command tells EFP not to redisplay the third error, but to display the fourth error, which applies to line 12 of your source code:

```
ERR-ØØ4 error message
   12.ØØØØ source line
1Ø1.ØØØØ ▷
```

The cursor is positioned to accept the next command – either an EDT command or another EFP command.

If EFP has already displayed all the errors and their corresponding source lines, EFP displays the message:

```
EFPØØ6 ALL ERRORS DISPLAYED FOR MODULE source-module-name
                                                or
                                       program-unit-name
     ▷
```

Again, the cursor is positioned to accept the next EFP or EDT command.

NOTE:

For FORTRAN IV programmers only: If your source module contains multiple program units and you specify the X parameter, EFP automatically goes on to the next program unit after it displays all the errors and source lines for one program unit. In that case, along with message EFP006, EFP also displays message EFP005, the error summary message for the next program unit.

program-unit-name

Applies only to FORTRAN IV source modules containing multiple program units. This parameter lets you specify to EFP that you want to see and correct errors for one specific program unit within your source module.

Notice that this parameter doesn't let you see errors for multiple source modules in your error file. To see errors for multiple source modules, you must use Format 2 of the EFP command.

NOTE:

If the name of the program unit is SUM, SUMM, SUMMA, SUMMAR, SUMMARY, SOURCE, SOU, SOUR, SOURC, or END, you must enclose the program unit name in parentheses.

`error-range`

Displays a range of errors along with their corresponding source lines. You can specify a single error, nonconsecutive errors, consecutive errors, or any combination. The conventions for specifying the error range are:

- Separate nonconsecutive error numbers by commas.
- Specify consecutive error numbers by giving the first and last errors separated by a colon.
- Specify error numbers in ascending order.
- Don't specify any error number greater than the number of errors EFP reports in message EFP005, which is the error summary message.

For example, let's say EFP reports 13 errors in your source code. If you specify @EFP 2:5,7,11 as your command, then EFP displays only errors 2 through 5, and errors 7 and 11. After it displays all the errors, EFP positions the cursor for you to correct the errors.

`'search-string'`

Lets you display, by error message text, only those errors of one specific type, along with the source lines that contain that type of error. The search string can't be longer than 50 characters.

For example, let's say you've already reviewed your error summary and viewed all the errors in your error file. You see that in the data division of your COBOL program, the same type of error occurs more than once, namely, periods missing in elements. If you specify:

```
@EFP 'PERIOD MISSING IN''
```

EFP searches the error file, finds that errors 9, 15, and 20 are that particular error and lines 16, 19, and 28 of your source code are the lines containing that problem. EFP then displays:

```
ERR-ØØØ9 'PERIOD MISSING IN element. PERIOD ASSUMED'
    16.ØØØØ         Ø2 NAME        PIC X(25)

ERR-ØØ15 'PERIOD MISSING IN element. PERIOD ASSUMED'
    19.ØØØØ         Ø2 STATE       PIC X(2)

ERR-ØØ2Ø 'PERIOD MISSING IN element. PERIOD ASSUMED'
    28.ØØØØ         Ø2 FILLER      PIC X(1) VALUE SPACES

1Ø1.ØØØØ ▷
```

EFP positions the cursor for you to enter an EDT command to correct each error individually. The cursor is at line 101 because your source module contains 100 lines. When you've corrected all the errors, you can enter another EFP command.

Some typical examples of Format 1 follow. They're based on two separate sample situations.

- These examples are based on the following sample situation. Your error file contains 20 errors for your COBOL source module. Here's a list of the source lines that contain some of the errors:

  - Error 1 applies to line 3 of your source code.
  - Error 2 applies to line 6 of your source code.
  - Error 4 applies to line 12 of your source code.
  - Errors 9, 15, and 20 are all the same type and apply to lines 16, 19, and 20 of your source code, respectively.

  @EFP
  Displays error 1 in your error file along with line 3 of your source code, even if you've already seen it

  @EFPX
  Displays error 2 in your error file along with line 6 of source code because you've already seen error 1

  @EFP 2:5,7,11
  Displays only errors 2 through 5 and errors 7 and 11, along with the source lines that contain those errors.

  @EFP 'PERIOD MISSING IN''
  Displays errors 9, 15, and 20, which are all the same error, along with source lines 16, 19, and 28, respectively, because those are the lines that contain those errors

@EFPX 2:5,7,11

Displays errors 2 through 5, 7, and 11, along with the source lines that contain them, but only if EFP hasn't already displayed them

@EFPX 'PERIOD MISSING IN'

Displays errors 9, 15, and 20, which are all the same error, along with source lines 16, 19, and 28, which contain that error, but only if EFP hasn't already displayed them

- This example is based on the sample situation in which the user's FORTRAN IV source modules contain multiple program units.

  @EFP PROGUA

  Displays errors for the specific program unit named PROGRUA in your source module

Format 2 (for FORTRAN IV compilations of multiple source modules):

```
@EFP SOURCE source-module-name,source-file-name,vsn
```

where:

`source-module-name`
Specifies the name of a specific source module in your FORTRAN IV compilation that you want to correct.

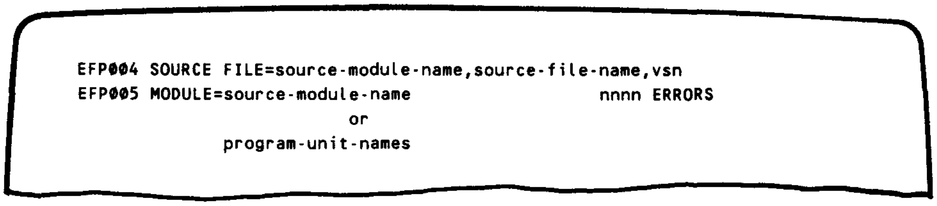
`source-file-name`
Specifies the name of the source file that contains the source module you're correcting.

`vsn`
Specifies the volume serial number of the volume containing that source file.

Before entering Format 2, first key in an @WRITE command to save the current contents of your EDT work space. Then enter an @DELETE command to empty the EDT work space. These items are necessary so EDT can synchronize the errors in the error file with source lines in your source module.

Once you enter the EFP SOURCE command, EFP then:

- identifies the source module for which the error file reports errors;
- scans the error file and reports an error summary to you at your workstation; and
- displays the following error summary screen:



```
EFPØØ4 SOURCE FILE=source-module-name,source-file-name,vsn
EFPØØ5 MODULE=source-module-name                    nnnn ERRORS
                           or
              program-unit-names
```

where:

EFPØØ4
: Is the message that identifies the information you supplied with the EFP SOURCE command.

EFPØØ5
: Identifies the number of errors EFP reports for the program units named.

Using the information in EFP004, EFP directs EDT to write your source file to the EDT work space, which lasts for the duration of the EDT session. Hereafter, you control your source module through regular EDT commands. And, you control your error file through the EFP commands.

A typical example of Format 2 follows:

@EFP SOURCE MODULE2,SOURCEFILEA,D00639

Displays errors for the source module (named MODULE2) in your FORTRAN IV source file (named SOURCEFILEA), which resides on the volume whose serial number is D00639

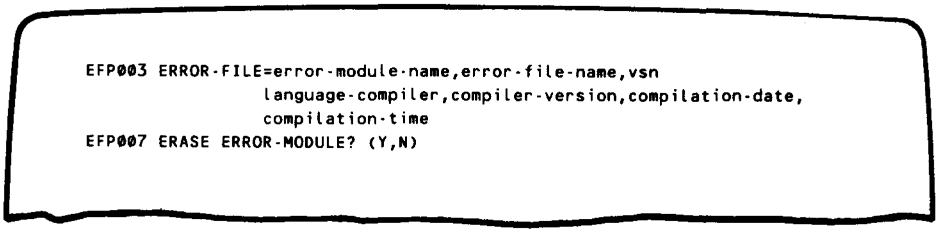
11.3.2.3. Terminating EFP (END)

After you've seen or corrected all the errors in your source code, the END command lets you terminate your EFP session. Of course, if you don't have time to correct all the errors in one session, you can terminate EFP and reactivate it later to finish correcting your source code.

The format of the END command is:

```
@EFP END
```

When you specify this command, EFP displays:



```
EFPØØ3 ERROR-FILE=error-module-name,error-file-name,vsn
                  language-compiler,compiler-version,compilation-date,
                  compilation-time
EFPØØ7 ERASE ERROR-MODULE? (Y,N)
```

where:

EFPØØ3
: Is the message that identifies the error file for your program, as well as header information that your language compiler supplies about the compilation of your program.

EFPØØ7
: Is the message that asks if you want to erase the module in your error file containing the errors for the module you're correcting. Respond Y for yes if you've seen and corrected all the errors EFP reported; respond N for no if you want to go back later and use EFP to finish correcting errors. (Note that EFP flags all errors as undisplayed once you terminate EFP).

After you respond to message EFP007, you can no longer use any EFP command. EFP erases the temporary work file that contained the error file for the duration of the EFP session. At this time, you must write the corrected source code from the EDT work space to a permanent source file, usually its original file. To do this, use the EDT @WRITE command. After you write the file back, you simply issue an EDT @HALT directive to terminate the entire EDT session.

11.4. SAMPLE EFP SESSION

In this sample EFP session, we will show you how to use EFP for correcting an RPG II source program. This program prints a listing of your employee staff and is named STAFPR. The job control stream that compiles your program includes parameters for an error file, therefore, you can use EFP to interactively correct compilation errors as soon as the compiler successfully compiles your program.

Here is the source code for your program:

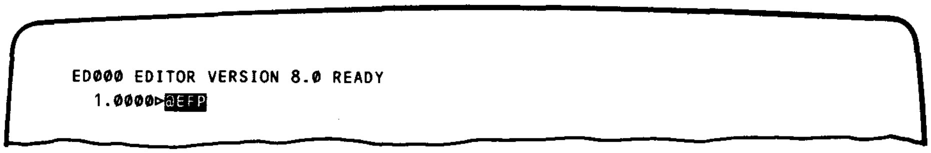
```
ØØ1  ØØ Ø1Ø H                                                            STAFPR
ØØ2  Ø1 Ø1Ø FINDATA  IPEAF  8Ø  8Ø             READER
ØØ3  Ø1 Ø2Ø FLIST    Ø   F 132 132      OF     PRINTER
ØØ4  Ø2 Ø1Ø IINDATA  AA  Ø1
ØØ5  Ø2 Ø2Ø I                                         1 8Ø INREC
ØØ6  Ø3 Ø1Ø C   Ø1      COUNT     ADD   1         COUNT  3Ø
ØØ7  Ø4 Ø1Ø OLIST   H   2Ø7   1P
ØØ8  Ø4 Ø2Ø O       OR        OF
ØØ9  Ø4 Ø3Ø O                                    28 'STAFF FILE LISTING'
Ø1Ø  Ø4 Ø4Ø O        D  2     Ø1
Ø11  Ø4 Ø5Ø O                         INRED      8Ø
Ø12  Ø4 Ø6Ø O        T  3     LR
Ø13  Ø4 Ø7Ø O                                    25 'TOTAL RECORDS LOADED='
Ø14  Ø4 Ø8Ø O                         COUNT Z    35
```

Notice that the record names on the input specification form on line 5 and the output specification form on line 11 don't match. (We've circled the mismatch.) Of course, in a real programming situation, you would be unaware of what errors your program contained prior to compiling it.

In this example, your entries are shown in white letters on a black background, while messages displayed on the screen are shown in black letters on white.

- Step 1

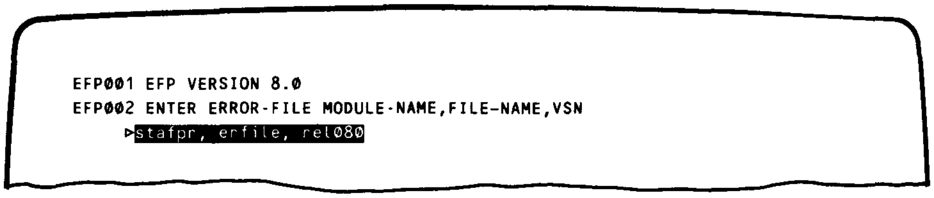
As soon as your system displays the message telling you that your compilation job terminated normally, you can use EFP to correct your programming errors. You activate EDT as described in Section 2 and enter an @EFP directive. Then you press the XMIT key.



```
ED000 EDITOR VERSION 8.0 READY
  1.0000▷@EFP
```

- Step 2

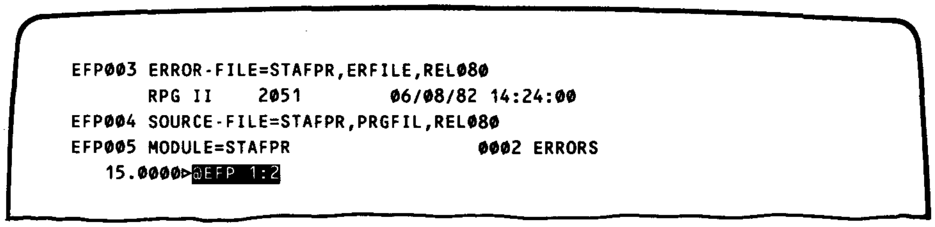
EFP displays a message telling you it's ready. It requests information about your error file, namely, its module and file name, both of which you assigned to the file when you wrote your compilation job control stream. It also asks for the volume serial number of the volume containing the error file. You key in the information in the order EFP requests and then press the XMIT key.



```
EFP001 EFP VERSION 8.0
EFP002 ENTER ERROR-FILE MODULE-NAME,FILE-NAME,VSN
     ▷stafpr, erfile, rel080
```

- Step 3

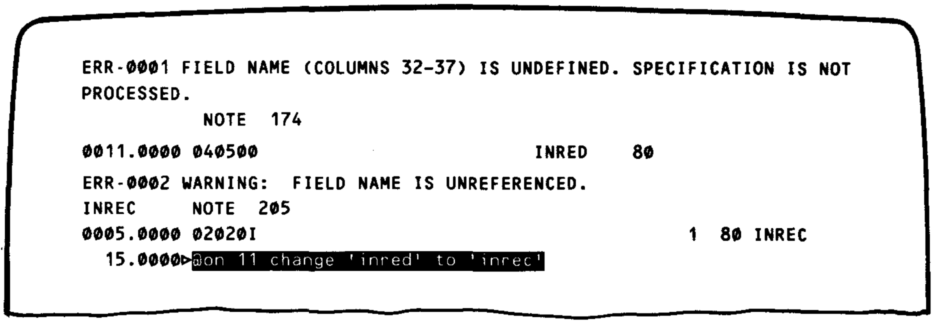
EFP displays the error summary screen for your source program. It consists of three messages–EFP003, EFP004, and EFP005 – and a cursor for you to enter a command. Message EFP003 tells the information abour your error file that you supplied in response to message EFP002 and information about the compilation of your program. Message EFP004 tells the module name, file name, and the volume serial number of the volume containing your source program. EFP005 tells the number of errors your error file holds for the source module (in this case, two errors). EFP copies your source module into EDT's work space and positions the cursor at line 15 because your source code is 14 lines long. You enter an EFP command to see both errors at one time. Then you press the XMIT key.



```
EFP003 ERROR-FILE=STAFPR,ERFILE,REL080
       RPG II    2051       06/08/82 14:24:00
EFP004 SOURCE-FILE=STAFPR,PRGFIL,REL080
EFP005 MODULE=STAFPR                  0002 ERRORS
   15.0000▷@EFP 1:2
```

- Step 4

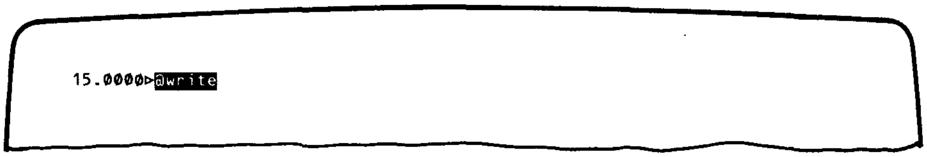
EFP displays both errors it holds for the source module, along with the lines that contain the errors. As you study the errors, you see that they are caused by the mismatch in the record name that we pointed out at the beginning of this sample session. The cursor is positioned at line 15 to accept your next command. You key in an @CHANGE command to correct the record name on line 11 and press the XMIT key.



```
ERR-0001 FIELD NAME (COLUMNS 32-37) IS UNDEFINED. SPECIFICATION IS NOT
PROCESSED.
            NOTE  174
0011.0000 040500                        INRED    80
ERR-0002 WARNING:  FIELD NAME IS UNREFERENCED.
INREC      NOTE  205
0005.0000 02020I                                  1  80 INREC
  15.0000▷@on 11 change 'inred' to 'inrec'
```

- Step 5

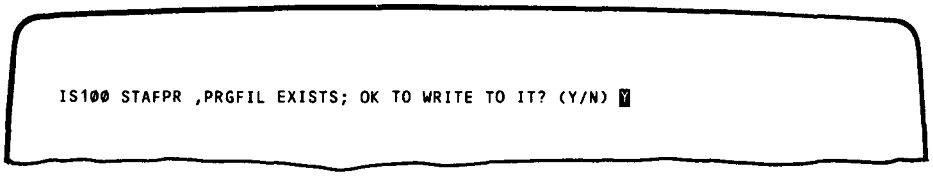
The cursor is again positioned on line 15 to accept your next command. You know that this correction will result in a clean compilation, so you key in an @WRITE command to save the corrected source code. You don't have to specify any parameters on your command; EDT will automatically write the contents of its work space to the module you're correcting. The you press the XMIT key.



```
15.0000▷@write
```

- Step 6

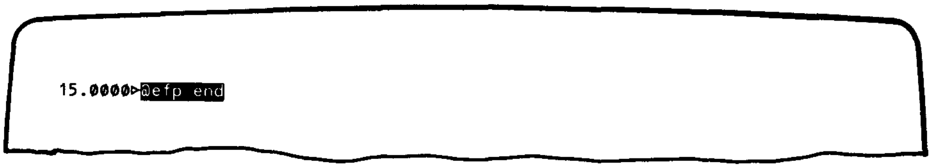
EDT reminds you that your source module already exists and double-checks if you want to overwrite it. You want to overwrite it so you respond Y for yes and press XMIT key.



```
IS1ØØ STAFPR ,PRGFIL EXISTS; OK TO WRITE TO IT? (Y/N) Y
```

- Step 7

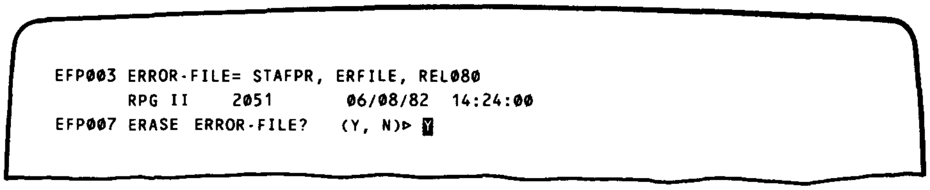
EDT positions the cursor on line 15 again to accept your next command. You enter an @EFP END command to end your EFP session and press the XMIT key.



```
15.ØØØØ▷@efp end
```

- Step 8

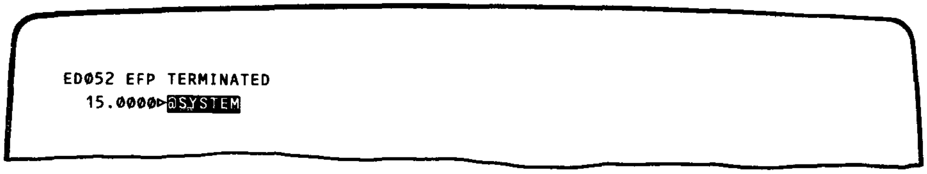
EFP redisplays message EFP003 telling information about your error file and the compilation of your source program. It also displays message EFP007, which asks if you want to erase the module in your error file containing the errors for the source module. You respond Y for yes because you're finished correcting your source module, and then you press the XMIT key.



```
EFPØØ3 ERROR-FILE= STAFPR, ERFILE, RELØ8Ø
       RPG II    2Ø51       Ø6/Ø8/82  14:24:ØØ
EFPØØ7 ERASE  ERROR-FILE?   (Y, N)▷ Y
```

- Step 9

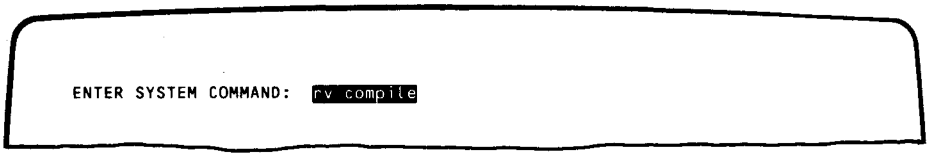
EFP terminates and EDT positions the cursor to accept your next command. You enter an @SYSTEM directive so you can recompile your program without exiting from EDT. Then you press the XMIT key.



```
EDØ52 EFP TERMINATED
  15.ØØØØ▷@SYSTEM
```

- Step 10

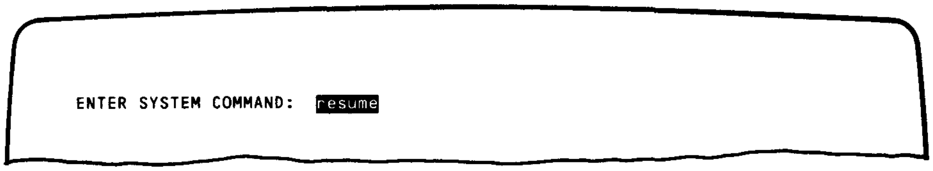
EDT displays a message telling you to enter a system command. You enter the run command to run your compilation job, then you press the XMIT key.



```
ENTER SYSTEM COMMAND: rv compile
```

- Step 11

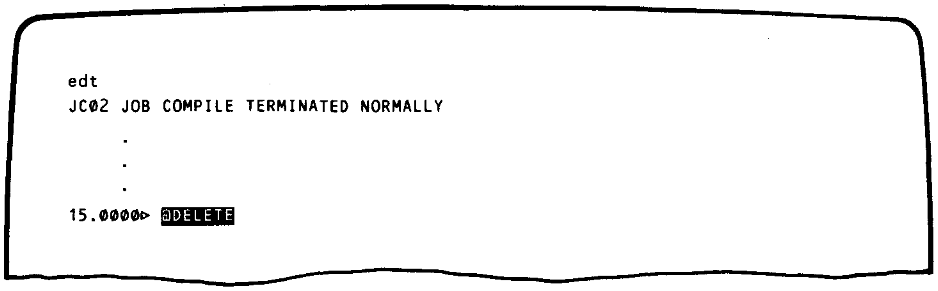
EDT displays another message telling you to enter a system command. You enter the RESUME command to continue with your EDT session and you press the XMIT key.



```
ENTER SYSTEM COMMAND: resume
```

- Step 12

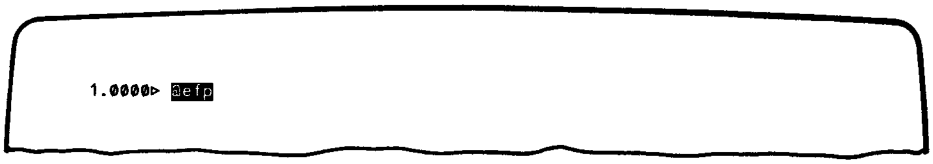
EDT positions the cursor at line 15 to accept another EDT command. At this point, you see SYS MSG displayed on the system state line. You press the FUNCTION and the SYS MODE keys simultaneously. On the top line of your screen you see your original system command, EDT, that activated EDT. On the second line, you see messages about the compilation of your program. You press the XMIT key after reading each one. When you see the message telling you that your compilation job terminated normally, you press the XMIT key once more and EDT returns the cursor to line 15 to accept your next command. Enter an @DELETE command to empty EDT's work space. Then press the XMIT key.



```
edt
JC02 JOB COMPILE TERMINATED NORMALLY
      .
      .
      .
15.0000> @DELETE
```

- Step 13

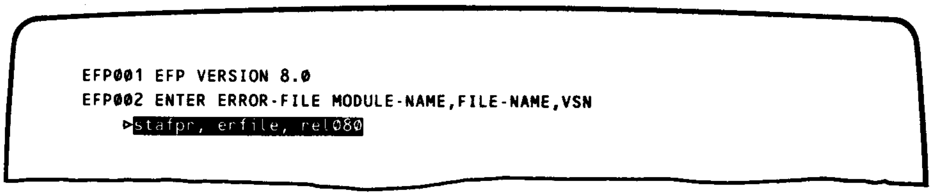
EDT empties its work space and positions the cursor on line 1 to accept your next command. You enter an @EFP directive to reactivate EFP. Then you press the XMIT key.



```
1.0000▷ @efp
```

- Step 14

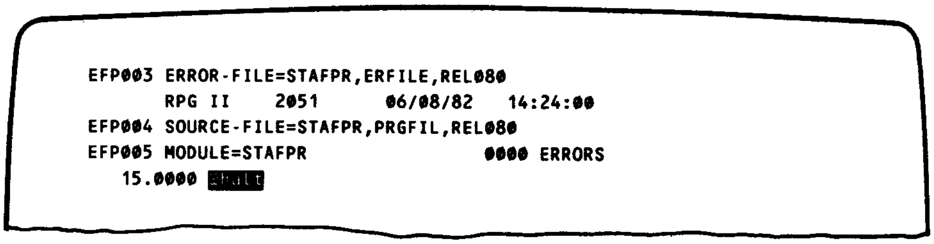
Again, EFP tells you it's ready and asks for the module and file name of your file and the volume serial number of the volume containing it. You key in the information and press the XMIT key.



```
EFP001 EFP VERSION 8.0
EFP002 ENTER ERROR-FILE MODULE-NAME,FILE-NAME,VSN
    ▷stafpr, erfile, rel080
```

- Step 15

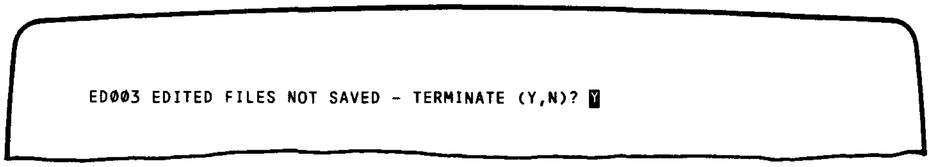
EFP again displays the error summary screen for your program. It shows that your error file contains no errors for your source code. So you issue an @HALT directive to end both your EFP and EDT sessions. Then you press the XMIT key.



```
EFP003 ERROR-FILE=STAFPR,ERFILE,REL080
       RPG II    2051      06/08/82   14:24:00
EFP004 SOURCE-FILE=STAFPR,PRGFIL,REL080
EFP005 MODULE=STAFPR                  0000 ERRORS
   15.0000 @halt
```

- Step 16

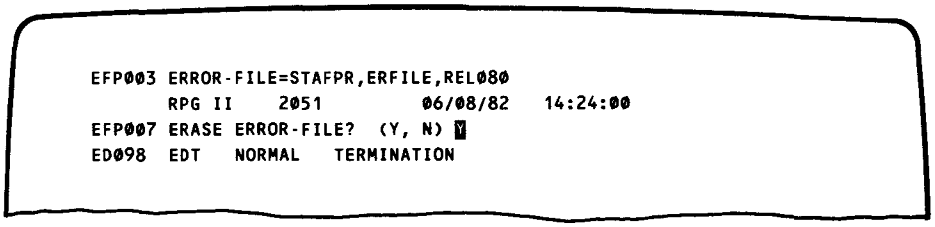
EDT reminds you that you didn't write the contents of its work space to a permanent file and asks if you still want to terminate the session. Because you made no changes to the source program in EDT's work space, you don't have to save a copy of it. So you respond Y for yes and press the XMIT key.



```
EDØØ3 EDITED FILES NOT SAVED - TERMINATE (Y,N)? Y
```

- Step 17

EFP then redisplays message EFP003 and EFP007. You respond Y for yes to message EFP007 because your program is already error free. Then you press the XMIT key and EDT automatically terminates both the EFP and EDT sessions.



```
EFPØØ3 ERROR-FILE=STAFPR,ERFILE,RELØ8Ø
       RPG II    2Ø51        Ø6/Ø8/82   14:24:ØØ
EFPØØ7 ERASE ERROR-FILE?  (Y, N) Y
EDØ98  EDT   NORMAL   TERMINATION
```

12. Error Detection and Recovery

There are several kinds of errors that may occur during an EDT session. These are:

- command errors;
- character string, line range, or column range errors;
- file errors; and
- workspace file errors.

Command errors occur when invalid commands are keyed in at the workstation. These errors may be detected as soon as the commands are keyed in or after the commands have begun to be processed. This, however, depends on the nature of the error. Unrecognizable commands and invalid command combinations are detected immediately, whereas invalid parameters in commands often are not detected. These errors are usually recoverable.

Errors also result when invalid character strings, line ranges, or column ranges are specified in a command. These errors are not detected until the command is processed up to that range or until the command is processed through the entire file. Consequently, a command may be partially completed before an error is detected. These errors are usually recoverable.

File errors result when the INPUT, READ, or WRITE commands are executed incorrectly. These errors are usually recoverable.

Work-space file errors occur when the contents of the EDT work-space file are lost or destroyed. These errors are usually unrecoverable. You should, however, attempt a retry to save the EDT work space. Operator intervention may also be required.

Generally, when an error message is displayed, EDT retypes the current line number on the next line. You may then key in the corrected version of the line.

The error messages displayed describe the situation that caused the error. They also suggest possible procedures to correct the errors.

A complete list of the EDT error messages appears in the OS/3 system messages programmer/operator reference.

Appendix A. Abbreviations

Table A-1 contains an alphabetized list of the minimum abbreviations recommended for the EDT commands, command options, keywords, command modifiers, EDT procedure file commands, EDT variable commands, directives, screen commands, and EFP commands. You must key in at least those characters shown, but you may also key in additional consecutive characters to improve readability.

Table A-1. Abbreviations

Term	Abbreviation	Term	Abbreviation	Term	Abbreviation
ALL	A	END	E	ON	ON
ASSIGN	AS	ENDCOL	EN	PARAMS	PA
ATSIGN	A	EXCLUDE	E	PRINT	P
BLOCK	BL	FIND	FIN	PROC	PRO
BY	B	FIRST	F	PROMPT	PROM
BUFFER	B	FORMAT	FORMAT	PUNCH	PU
CHANGE	C	FSTATUS	FS	READ	R
CHAR	C	GOTO	G	REMOVE	REM
CHECK	CHE	HALT	H	RESTORE	RES
CLEAR	CL	HELP	HE	RETURN	RET
COBOL	COB	IF	IF	ROLL	RO
COLON	CO	INPUT	INP	RPG	RPG
COLUMN	COL	INSERT	I	SEQUENCE	SEQ
CONTINUE	CON	KEY	K	SET	S
COPY	CO	KKEY	KK	SHOW	SH
DELETE	D	LINE	L	STRIKE	ST
DISPLAY	DI	LIST	L	STRIP	S
DO	DO	MOVE	M	SYSTEM	SY
DROP	DR	NOCHANGE	N	TABS	T
EFP (directive)	EFP	NOP	NOP	TO	T
EFP (command)	EF	NOPRINT	N	UPDATE	U
EFP SOURCE	EF SOU	NOT	NOT	WIDTH	W
EFP END	EF END	NUMBER	NU	WRITE	W
EFP SUMMARY	EF SUM	OFF	OFF		

Appendix B. Command Processing Loop

The following outline describes the order in which EDT processes a command line. It represents the loop through which commands and modifiers are processed.

1. Input

   a. NUMBER command and data line are read from workstation.

   b. READ and FSTATUS commands are read from permanent file. KEY, KKEY, and SHOW options may alter the scope of read.

   c. All other commands are read from work space. Line range may limit scope of read.

2. Search string processing (FIND, REMOVE, COPY, UPDATE, CHANGE, MOVE, PUNCH, PRINT, LIST)

   If a search string is specified, it must be found in the line if the line is to be processed. The COLUMN modifier limits the search for a search string in the line to a specified range. If a search string is specified but not found in the line, processing returns to the beginning of the loop to get the next input line.

3. Change string processing (CHANGE, INSERT, NUMBER, SEQUENCE, REMOVE)

   If a change string is specified or implied (REMOVE) changes are made now. If the ALL modifier is specified, all occurrences of the search string are changed to the change string, but if ALL is omitted, only the first change in the line is processed.

4. Update processing (UPDATE)

   The line is now displayed for updating if UPDATE was specified. If a command is entered at this point, the loop may be terminated or processing is altered.

5. Delete processing (DELETE, MOVE)

   The line is deleted from the input location of the work space and will not be rewritten if it has been changed.

6. Output to work space

   a. All changed lines are written back to the work space.

   b. Copy lines (COPY, MOVE) are written to the destination.

7. Show processing (SHOW)

   The line is reformatted according to the specifications on the SHOW option.

8. Output (WRITE, PUNCH, LIST, PRINT)

   The line is output to a permanent file (WRITE), a spool file (PUNCH, LIST), or the workstation (PRINT).

9. End-of-file processing (FIRST, SEQUENCE, NUMBER)

   This represents the end of the processing loop. The sequence string is incremented and the FIRST option is tested. If the FIRST option is specified, the loop terminates and end-of-file processing occurs. Otherwise, the loop is reentered to process the next input line.

Appendix C. Summary of Command I/O Operations and Options

Table C-1 shows the input and output locations and the ranges and strings that are permitted in each command.

Table C—1. I/O Operations and Options for Commands

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
CHANGE	X				X					X	▨		X	X		
COPY	X				X					X		▨	X	X		
DELETE	X			X						X			X	X		
FIND	X									▨			X	X		
FSTATUS			X		X					X				X	X	
INSERT	X				X						▨		X	X		
LIST	X							X		X			X	X	X	
MOVE	X			X	X					X		▨	X	X		
NUMBER			X		X						▨					X
PRINT	X						X			X			X	X	X	
PUNCH	X							X		X			X	X	X	
READ		X			X					X				X	X	
REMOVE	X				X					▨			X	X		
SEQUENCE	X				X					X	▨		X	X		X
UPDATE	X				X				X	X			X	X		
WRITE	X					X				X			X	X	X	
data			X		X											

LEGEND:

A	Source is work-space file	G	Output is workstation	M	Line range allowed
B	Source is permanent file	H	Output is spool file	N	Column range allowed
C	Source is workstation	I	Update command	O	Show range allowed
D	Source is to be deleted	J	Search string allowed	P	By increment allowed
E	Output is work-space file	K	Change string allowed	X	Permitted
F	Output is permanent file	L	Destination allowed	▨	Required

Appendix D. Comparison of OS/3 General Editor and BEM Editor

The OS/3 general editor (EDT) is actually an extension of the BEM editor (BEM EDT) used in Series 90. However, there are several differences between the two editors, which will be discussed here.

- The type of files accessed by EDT is more versatile than BEM EDT.

  EDT can access SAT and MIRAM library files, MIRAM data files, spool files and sequential device files (tape, card and diskette files), whereas BEM EDT can access only SAT library files.

- EDT uses the colon (:) as the column and line range separator, whereas BEM EDT uses the hyphen (-).

  The colon has greater flexibility in OS/3 expression evaluation. Note that EDT enables you to change the column and line range separator from the preset colon to another symbol via the @SET COLON=range-separator command.

- EDT recognizes #G as the symbol for global variables. BEM EDT uses #.

  The use of #G as the global variable makes EDT easier to use since it eliminates the need to specify ## for each # when doing a search operation.

  For example, to change the string IF# TO IF, BEM EDT uses the statement

  ```
  @ON & C 'IF##' TO 'IF'
  ```

  while EDT uses this statement

  ```
  @ON & C 'IF#' TO 'IF'
  ```

- EDT indicates the start column of a found search string by an opening bracket ([). BEM EDT uses a percent sign (%). EDT uses a closing bracket (]) to indicate the end column of a found search string, whereas BEM EDT has no special symbol for the end column of a found search string.

  The use of the symmetric symbols [and] gives EDT increased functionality not available in BEM EDT.

- EDT uses #Gn(x:y) (where x:y denotes the column range) to define a variable substring. BEM EDT uses #Gn(s,L) (where s denotes the starting column and L denotes the length of the substring).

  For example:

  1. `@ASSIGN G6='ABCDEFGH'`
  2. `@ASSIGN G9='#G6(3:5)`
  3. `@ASSIGN G9='#G6(3,3)`

  Executing 1 and 2 in EDT is equivalent to executing 1 and 3 in BEM EDT. They both assign the value CDE to #G9.

- EDT recognizes n(x:y) as the syntax for a line substring, while BEM EDT uses n:x-y.

  For example:

  1. `ABCDEFGHIJKL`
  2. `@ASSIGN G1=1`
  3. `@ASSIGN G2=#G1(5:7)`
  4. `@ASSIGN G2=#G1:5-7`

  Executing 1, 2, and 3 in EDT is equivalent to executing 1, 2, and 4 in BEM EDT. They both assign the value EFG to #G2.

- The current line number and increment are local for EDT but are not in BEM EDT.

  EDT saves the current line number and increment for each EDT procedure file. BEM EDT saves the current line number but always resets the increment to 1 when entering or returning from a procedure file.

- EDT (using a freeform scan) recognizes as command lines those lines whose first nonblank character is a command trigger. BEM EDT recognizes the command trigger only in column 1, thus restricting flexibility in structuring commands.

  EDT allows spaces to precede the command trigger and relieves the user of worrying whether the command trigger was entered in column 1; thus, it is easier to use EDT.

- EDT supports variable-length records and fixed-length records, whereas BEM EDT supports only fixed-length records.

  EDT recognizes records up to the current buffer size (128<buffer<2048) and optionally strips trailing spaces and X'00's. BEM EDT considers all records to be 128 bytes.

  The use of variable-length records in EDT allows for expanded record sizes as well as more efficient space use for records less than 128 bytes.

- EDT and BEM EDT default differently when there is no print option specified on the DO command.

  EDT defaults to the print option of the procedure file that called it or to NOPRINT if it is called from the main work file. BEM EDT defaults to NOPRINT. EDT allows for greater control and flexibility with the PRINT and NOPRINT options on the DO command.

- EDT recognizes and supports the following function keys: F1, F2, F3, F4, F5, F6, F12, F13, F14, F15 (EOF), F18, and F19. BEM EDT does not recognize function keys; however, it does have the same functionality available as F2. There is no equivalent for EDT's F1.

  EDT has more interrupt capability than BEM EDT through the use of function keys.

- The following BEM EDT commands are not supported by EDT:

  1. @HELP (Note that the @HELP screen command (Section 8) is not the same as the BEM @HELP command.)

  2. @RSP

  3. @UPPER

  4. @LOWER

  5. @SET PAGE=

  6. @TYPE

  7. @DESEQ

  Commands 2 and 5 have no equivalent functionality in EDT. Commands 1, 3, 4, and 6 have equivalent system commands that can be used. EDT's keyword KKEY has approximately the same functionality as 7, but is more thorough in syntax checking.

Appendix E. Positional Parameters for the READ and WRITE Commands

The READ and WRITE (file) parameters may be entered in a positional format, instead of the keyword formats explained in 3.14 and 3.18. In a positional format, only the actual parameters for MODULE, FILENAME, VSN, and TYPE are entered; keywords are not used. The positional parameters are separated from each other by a comma and from the command by a space. The positional parameter format permits compatibility with the SPERRY UNIVAC BEM Editor. Four parameters may be entered in the positional format in the following order: module-name, filename, volume, module-type. All positional parameters are optional; however, if you do not enter the module name, but do enter the file name and volume (volume-serial-number), the file name must be enclosed in quotation marks (e.g., "filename").

Any time you omit a parameter, you still must include a comma to serve as its placeholder.

The following is an example of a positional parameter string in the READ command:

```
@READ PAYJOB1,PAYFILE,ABC466,S,DEV=DISKETTE
```

This command reads a source module named PAYJOB1 that resides in a file named PAYFILE on diskette volume ABC466 to the current work-space file.

The following is an example of a positional parameter string in the WRITE command:

```
@WRITE "PAYROLL",ABC466,DEV=DISK
```

This command writes a copy of the current work-space file to the file named PAYROLL on disk volume ABC466.

Appendix F. Command Summary

F.1. SUMMARY OF EDT COMMANDS

The formats and explanations for the EDT commands are summarized in Table F–1. The commands are listed in alphabetical order.

Table F–1. EDT Command Summary (Part 1 of 6)

Command	Format	Explanation
@	$@\left\{\begin{array}{l}\texttt{line-number [increment]}\\ +\\ -\end{array}\right\}\left[:\left\{\begin{array}{l}\texttt{data}\\ \texttt{command}\end{array}\right\}\right]$	Sets the current line number and increment for data and command lines keyed in at the workstation
CHANGE	`@C ['search-string'[*n]] TO 'change-string'[*n]`	Replaces an existing string in the current work-space file with a new string

COPY	`@CO [line-range]['search-string'[*n]]`
	`TO destination`
		Copies lines in the current work-space file to new line locations without deleting the original lines

DELETE	`@D [line-range]['search-string'[*n]]`	Erases specified lines from the current work-space file
FIND	`@FIN 'search-string'[*n]`	Locates the first occurrence of a string in the work-space file and assigns its corresponding line number to the variable ? and the column numbers of the first and last columns it occupies to [and] respectively

Table F-1. EDT Command Summary (Part 2 of 6)

Command	Format	Explanation
FSTATUS	To specify file parameters for any file for which you want a list of modules, use this format: @FS[MODULE=module-name] [,TYPE={module-type S}] ,FILENAME={filename 'filename' "filename"} [,RDPASS=password] ,VSN=volume [,DEVICE={did DISK DISKETTE}]	Creates in the work-space file a list of all modules contained in a specified program library
INSERT	@I 'change-string'[*n]	Inserts a specified string into lines in the current work-space file
LIST	@L [line-range]['search-string'[*n]][IMMEDIATE]	Prints specified lines from the current work-space file on the printer
MOVE	@M [line-range]['search-string'[*n]] TO destination	Transfers specified lines to new line locations in the work-space file and deletes the original lines and line numbers
NUMBER	@NU 'sequence-string'[*n] [BY increment]	Inserts sequence numbers into input lines
PRINT	@P [line-range]['search-string'[*n]]	Displays specified lines from the current work-space file on the workstation screen
PUNCH	@PU [line-range]['search-string'[*n]][IMMEDIATE]	Reproduces specified lines from the current work-space file on cards
READ	To read a SAT or MIRAM library module from disk or format label diskette to the current work-space file, use this format: @READ MODULE=module-name [,TYPE={module-type S}] [,TRUNC={YES NO}],FILENAME={filename 'filename' "filename"} [,RDPASS=password],VSN=volume [,DEVICE={did DISK DISKETTE}] △[{KEY=start-col-no:end-col-no KKEY=start-col-no:end-col-no SHOW△first-col-no:last-col-no}]	Reads a copy of a library module or program library into the work-space file

Table F-1. EDT Command Summary (Part 3 of 6)

Command	Format	Explanation

READ
(cont)
	To read a MIRAM data file from disk or format label diskette to the current work-space file, use this format:
	@READ FILENAME={filename }[,RDPASS=password]
	{'filename'}
	{"filename"}
	,VSN=volume[,KEYNO={n}][,DEVICE={did }]
	{0} {DISK }
	{DISKETTE}
	[,BFSZ=n][,TRUNC={YES}]
	{NO }
	Δ[{KEY=start-col-no:end-col-no }]
	{KKEY=start-col-no:end-col-no }
	{SHOWΔfirst-col-no:last-col-no}
	To read a unit record file from a data set label diskette or from the card reader, use this format:
	@READ FILENAME={filename },VSN=volume
	{'filename'}
	{"filename"}
	,DEVICE={did }[,TRUNC={YES}]
	{DISKETTE} {NO }
	{RDR }
	Δ[{KEY=start-col-no:end-col-no }]
	{KKEY=start-col-no:end-col-no }
	{SHOWΔfirst-col-no:last-col-no}
	To read a file from a tape, use this format:
	@READ FILENAME={filename }[,RDPASS=password]
	{'filename'}
	{"filename"}
	,VSN=volume,DEVICE={did }[,BKNO={YES}]
	{TAPE} {NO }
	[,TRUNC={YES}]Δ[{KEY=start-col-no:end-col-no }]
	{NO } {KKEY=start-col-no:end-col-no }
	{SHOWΔfirst-col-no:last-col-no}	

Table F–1. EDT Command Summary (Part 4 of 6)

Command	Format	Explanation
READ (cont)	To read a file from the spool file to the current work-space file, use this format: @READ [JOB=jobname] [,HOLD={L N Y}] [,FILENAME={filename 'filename' "filename"}] [,ACCT=acct-no] ,QUEUE={LOG PRINT PUNCH RDR} [,ALL={YES NO}] [,SKIP={n 0}] [,TRUNC={YES NO}] Δ [{KEY=start-col-no:end-col-no KKEY=start-col-no:end-col-no SHOWΔfirst-col-no:last-col-no}] To read the same module or file last accessed through a previous @READ or @WRITE command, use this format: @READ To read the same module or file last accessed through a previous @READ or @WRITE command but read now with a KEY, KKEY, or SHOW parameter or any valid EDT command specified, use this format: @READΔ;Δ[{KEY=start-col-no:end-col-no KKEY=start-col-no:end-col-no SHOWΔfirst-col-no:last-col-no}] [valid EDT command]	
REMOVE	@REM 'search-string'[*n]	Deletes a specified string from lines in the work-space file
SEQUENCE	@SEQ {'sequence-string'[*n] *} BY increment	Inserts sequence numbers into existing lines in the current work-space file
UPDATE	@U [line-range]['search-string'[*n]]	Displays specified lines from the work-space file one at a time for you to edit or change

Table F-1. EDT Command Summary (Part 5 of 6)

Command	Format	Explanation
WRITE	To write the current work-space file to a SAT or MIRAM library module on a disk or format label diskette, use this format: @WRITE MODULE=module-name [,TYPE={module-type}] S ,FILENAME={filename }[,WRPASS=password] 'filename' "filename" [,DEVICE={did }] DISK DISKETTE ,VSN=volume [,CONTIG={YES}] [,INC={n}][,RCSZ=n][,SIZE=n] NO 1 [,SAT={YES}] NO To write the current work-space file to a MIRAM data file on a disk or format label diskette, use this format: @WRITE FILENAME={filename }[,WRPASS=password] 'filename' "filename" ,VSN=volume [,CONTIG={YES}] [,DEVICE={did }] NO DISK DISKETTE [,INC={n}] [,INIT={YES}] [,EXTEND={YES}] 1 NO NO [,KEYi={start-col-no:end-col-no }] (start-col-no:end-col-no, {DUP }, {CHG }) NDUP NCHG ,SIZE=n [,RCB={YES}] [,RCFM={FIX}] ,RCSZ=n NO VAR [,SCSZ={n }] [,BFSZ=n] 256 To write the current work-space file to a unit record file (i.e., to the printer, card punch, or to a data set label diskette), use this format: @WRITE FILENAME={filename },VSN=volume 'filename' "filename" ,DEVICE={did } [,RCFM={FIX}][,RCSZ=n] DISKETTE VAR PRINT PUNCH	Writes a copy of the current work-space file to: a program library or data file on disk, diskette, or tape, or to the spool file

Table F–1. EDT Command Summary (Part 6 of 6)

Command	Format	Explanation

WRITE (cont)	To write the current work-space file to a tape, use this format:
	
	@WRITE FILENAME={filename }[,WRPASS=password]
	{'filename'}
	{"filename"}
	
	,VSN=volume,DEVICE={did }[,BFSZ=n] [,BKNO={YES}]
	{TAPE} {NO }
	
	[,RCFM={FIXUNB}] [,RCSZ=n] [,INIT={YES}]
	{FIXBLK} {NO }
	{VARUNB}
	{VARBLK}
	{UNDEF }
	
	[,EXTEND={YES}]
	{NO }
	
	To write the current work-space file to the spool file, use this format:
	
	@WRITE [JOB=jobname][,HOLD={YES}]
	{NO }
	
	[,FILENAME={filename }] [,ACCT=acct-no]
	{'filename'}
	{"filename"}
	
	,QUEUE={PRINT}[,COPIES={n}]
	{PUNCH} {1}
	{RDR }
	
	To write to the same module or file last accessed through a previous @READ or @WRITE command, use this format:
	
	@WRITE
	
	To write to the same module or file last accessed through a previous @READ or @WRITE command, but written to now with any valid EDT command specified, use this format:
	
	@WRITEΔ;Δvalid EDT command	

F.2. SUMMARY OF EDT PROCEDURE FILE COMMANDS

The formats and explanations for the EDT procedure file commands are summarized in Table F–2. The commands are listed in alphabetical order.

Table F–2. Procedure File Command Summary

Command	Format	Explanation
DO	@DO proc-number $\left[\left\{\begin{array}{l}\text{PRINT}\\ \text{NOPRINT}\\ \text{REVERT}\end{array}\right\}\right]$	Executes a procedure file
END	@E	Terminates procedure file definition
GOTO	@G $\left\{\begin{array}{l}\text{label}\\ \text{line}\end{array}\right\}$	Permits branching within a procedure file
INPUT	@INP file-parameters $\left[\left\{\begin{array}{l}\text{PRINT}\\ \text{NOPRINT}\\ \text{REVERT}\end{array}\right\}\right]$	Loads and executes a procedure file
NOP	@NOP [comment]	Enters extra lines for branching or comments into a procedure file
PROC	@PRO [proc-number]	Begins procedure file definition
RETURN	@RET	Terminates procedure file execution

F.3. SUMMARY OF EDT VARIABLE COMMANDS

The formats and explanations for the EDT variable commands are summarized in Table F-3. The commands are listed in alphabetical order.

Table F-3. Variable Command Summary

Command	Format	Explanation
ASSIGN	@AS Gn= { 'string'[*n] n(x:y) n[±m] Gm LEN(n) }	Assigns values to EDT variables
DISPLAY	@DI { 'string'[*n] n(x:y) n[±m] Gm LEN(n) }	Displays a specified expression or the value of a specified expression from the work-space file on the workstation screen
IF	@IF.condition.command or @IF expression relation expression command	Permits an EDT command or EDT procedure file command to be executed based on some condition

F.4. SUMMARY OF EDT DIRECTIVE COMMANDS

The format and explanations for the directive commands are summarized in Table F–4. The commands are listed in alphabetical order.

Table F–4. Directive Summary (Part 1 of 2)

Command	Format	Explanation
CHECK	@CHE [{ON OFF}]	Determines if processed lines are to be displayed on the workstation screen
COBOL	@COB	Activates the COBOL editor
DROP	@DR	Deletes all lines in the entire EDT work-space file
EFP	@EFP	Activates the error file processor
FORMAT	@FORMAT parameter string (for RPGEDT) @FORMAT (for COBEDT)	Used only in conjunction with either RPGEDT or COBEDT. See the appropriate subeditor manual for information on the @FORMAT directive.
HALT	@H	Terminates the EDT session
RPG	@RPG	Activates the RPG II editor
SET	@S [CHAR=tab-character,TABS={columns OFF}] [,LINE=length][,EXCLUDE={exclusion-character OFF}] [,ATSIGN=command-trigger][,COLON=range-separator] [,ENCOL=end-column][,BUFFER={record-size OFF}] [,WIDTH=device-size][,CLEAR][,STRIP={ON OFF}] [,DISPLAY] [,SCRDSPLY={TRUNCATE FOLD}] [,ROLL={15(if SCRDSPLY=TRUNCATE) 8(if SCRDSPLY=FOLD) 1-15}] [,MODE={LINE SCREEN}][,LANGUAGE={FREEFORM FORTRAN COBOL RPG}] [,RECENTRY={SINGLE MULT}][,SCRFORM={UNDERLINE BLANK}]	Defines various parameters to EDT that collectively make up your EDT environment

Table F–4. Directive Summary (Part 2 of 2)

Command	Format	Explanation
SYSTEM	@SY [workstation-command]	Permits workstation commands to be issued during an EDT session or temporarily returns you to system mode

F.5. SUMMARY OF EDT SCREEN COMMANDS

Table F–5 summarizes the formats and explanations for the screen commands. It lists the commands in alphabetical order.

Table F–5. Screen Command Summary

Command	Format	Explanation
BLOCK	@BL	Displays a freeform screen that allows you to switch to block mode for entering multiple commands or data
HELP	@HE [error message code]	Displays help screens for any EDT error messages
PARAMS	@PA	Displays a screen showing the parameters on the @SET directive (those that make up your EDT environment)
PROMPT	@PROM [edt command]	Displays the EDT command menu screen or help screens for any of the EDT commands (meaning EDT commands, modifiers, directives, procedure file commands, variables, and screen commands)
RESTORE	@RES	Returns you to the point in your EDT session where you originally entered a screen command
ROLL	@RO	Displays freeform screens, showing the EDT work-space file, where you can update lines or simply view them

F.6. SUMMARY OF EFP COMMANDS

Table F–6 summarizes the formats and explanations for the EFP commands. It lists the commands in alphabetical order, rather than the order you might use them in.

Table F–6. EFP Command Summary

Command	Format	Explanation
EFP	To correct and display COBOL and RPG II errors and FORTRAN IV errors for one source module at a time, use: @EF[X]Δ[program-unit-name]Δ [error-range]Δ ['search-string'] To correct and display FORTRAN IV errors for compilations that process multiple source modules, use: @EF SOU source-module-name, source-file-name, vsn	Displays errors in your error file along with the source lines that contain those errors. Note that EFP is both an EDT directive and an EFP command.
END	@EF END	Terminates the error file processor
SUMMARY	@EF SUM	Displays an error file summary for the module you're correcting

Appendix G. Sample EDT Sessions

G.1. CREATING A MIRAM DATA FILE IN LINE MODE

The following example shows you how to create a MIRAM data file using EDT in line mode. Here, we activate EDT, enter our data, store our file as a MIRAM data file, and then terminate EDT.

```
Activates EDT                    EDT
                                ⎧1.ØØØØ▷JO MAXWELL,22
                                ⎪2.ØØØØ▷KEN FITZPATRICK,23
                                ⎪3.ØØØØ▷KATHY HOLIDAY,32
Our data                        ⎨4.ØØØØ▷MICHAEL MAST,35
                                ⎪5.ØØØØ▷SHARON PHILLIP,28
                                ⎪6.ØØØØ▷LARRY BERG,29
                                ⎪7.ØØØØ▷SARA KAFER,51
                                ⎩8.ØØØØ▷RAY NEUMANN,56
Stores data as MIRAM file        9.ØØØØ▷@WRITE FIL=DATAFILE,VSN=RELØ7Ø,SI=2
Terminates EDT                   9.ØØØØ▷@HALT
```

G.2. UPDATING A MIRAM DATA FILE IN LINE MODE

The following example shows you how to update a MIRAM data file using EDT in line mode. Here, we activate EDT, read in a copy of our file to the work-space file, and then display our file. Next, we make our changes, redisplay the file (to ensure that the changes have been made) and then store the new version of our file (overwriting the previous version). Last, we terminate the EDT session.

```
Activates EDT                    EDT
Transfers a copy of our file     1.ØØØØ▷@READ FIL=DATAFILE,VSN=RELØ7Ø
to the work-space file
Displays the work-space          9.ØØØØ▷@PRINT
file
                                ⎧1.ØØØØ▷JO MAXWELL,22
                                ⎪2.ØØØØ▷KEN FITZPATRICK,23
                                ⎪3.ØØØØ▷KATHY HOLIDAY,32
                                ⎪4.ØØØØ▷MICHAEL MAST,35
Our file                        ⎨5.ØØØØ▷SHARON PHILLIP,28
                                ⎪6.ØØØØ▷LARRY BERG,29
                                ⎪7.ØØØØ▷SARA KAFER,51
                                ⎩8.ØØØØ▷RAY NEUMANN,56
```

(continued)

```
Changes 23 to 24 on line 2      9.0000▷@ON 2 CHANGE '23'TO'24'
of the work-space file
Deletes lines 3 and 4           9.0000▷@DELETE 3:4
Displays our updated file       9.0000▷@PRINT
                                1.0000▷JO MAXWELL,22
                                2.0000▷KEN FITZPATRICK,24
Our updated file                5.0000▷SHARON PHILLIP,28
                                6.0000▷LARRY BERG,29
                                7.0000▷SARA KAFER,51
                                8.0000▷RAY NEUMANN,56
Writes our updated file to      9.0000▷@WRITE FILDATAFILE,VSN=REL070
A MIRAM file on disk
Reminds us that this file       IS100 FILE/MODULE ALREADY EXISTS;
already exists, and asks if     OK TO WRITE:TO IT? (Y,N)▷Y
we want to overwrite it. We
select Y, causing the previous
version of our file to be
overwritten.
Terminates EDT                  9.0000▷@HALT
```

G.3. CREATING A USER PROGRAM (SOURCE MODULE) IN LINE MODE

The following example shows you how to create a FORTRAN IV program using EDT in line mode. Here, we activate EDT, enter our source statements, store our program (as a source module in a program library), and then terminate EDT.

```
Activates EDT                    EDT
                                 1.0000▷C COMPUTE AVERAGE OF NUMBERS
                                 2.0000▷       KOUNT=0
                                 3.0000▷       TOTAL=0
                                 4.0000▷    30 READ(1,10) VALUE
                                 5.0000▷    10 FORMAT (G10.0)
                                 6.0000▷       IF(VALUE .GT.9.OE8) GO TO 20
Source statements                7.0000▷       TOTAL=TOTAL+VALUE
                                 8.0000▷       KOUNT=KOUNT+1
                                 9.0000▷       GO TO 30
                                10.0000▷    20 XMEAN=TOTAL / KOUNT
                                11.0000▷       WRITE(3,40) XMEAN
                                12.0000▷    40 FORMAT(' AVERAGE VALUE= ',F10.5)
                                13.0000▷       STOP
                                14.0000▷       END
Lists our file on printer       15.0000▷@LIST
Stores our program as a SAT     15.0000▷@WRITE MO=MYPROG,FIL=MYPROGFILE,
file on disk                            VSN=REL070,SAT=YES,SI=5
Terminates EDT                  16.0000▷@HALT
```

G.4. UPDATING A USER PROGRAM (SOURCE MODULE) IN LINE MODE

The following example shows you how to update a FORTRAN IV program using EDT in line mode. Here, we activate EDT and then read in a copy of our source program to the work-space file and display it. Next, we make our changes, display the changed statements, and then store the new version of our program (overwriting the previous version). Last, we terminate EDT.

```
Activates EDT                    EDT
Transfers a copy of our           1.ØØØØ▷@READ MO=MYPROG,FIL=MYPROGFILE,
file to the work-space file               VSN=RELØ7Ø
Displays the work-space file     15.ØØØØ▷@PRINT
                                 ( 1.ØØØØ▷C COMPUTE AVERAGE OF NUMBERS
                                 | 2.ØØØØ▷      KOUNT=Ø
                                 | 3.ØØØØ▷      TOTAL=Ø
                                 | 4.ØØØØ▷   3Ø READ(1,1Ø)VALUE
                                 | 5.ØØØØ▷   1Ø FORMAT (G1Ø.Ø)
                                 | 6.ØØØØ▷      IF(VALUE.GT.9.ØE8)GO TO 2Ø
                                 | 7.ØØØØ▷      TOTAL=TOTAL+VALUE
Our file                         { 8.ØØØØ▷      KOUNT=KOUNT+1
                                 | 9.ØØØØ▷      GO TO 3Ø
                                 |1Ø.ØØØØ▷   2Ø XMEAN=TOTAL / KOUNT
                                 |11.ØØØØ▷      WRITE(3,4Ø)XMEAN
                                 |12.ØØØØ▷   4Ø FORMAT(' AVERAGE VALUE= ',FØ.5)
                                 |13.ØØØØ▷      STOP
                                 (14.ØØØØ▷      END
Enters new data line at          15.ØØØØ▷@1Ø.5:C PRINT AVERAGE AND TERMINATE
line 10.5
Enters new data line at          1Ø.6ØØØ▷@1.5:C INITIALIZE COUNT AND TOTAL
line 1.5
Displays lines 1.5 and 10.5       1.6ØØØ▷@PRINT;1.5,1Ø.5
                                  1.5ØØØ▷C INITIALIZE COUNT AND TOTAL
                                 1Ø.5ØØØ▷C PRINT AVERAGE AND TERMINATE
Writes our updated file to        1.6ØØØ▷@WRITE MO=MYPROG,FILE=MYPROGFILE,
a SAT file on disk                        VSN=RELØ7Ø,SAT=YES
Reminds us that this file         IS1ØØ FILE/MODULE ALREADY EXISTS;
already exists, and asks if       OK TO WRITE TO IT? (Y,N)▷Y
we want to overwrite it. We
select Y, causing the pre-
vious version of our program
to be overwritten.
Terminates EDT                    1.6ØØØ▷@HALT
```

G.5. CREATING A JOB CONTROL STREAM IN LINE MODE

The following example shows you how to create a job control stream using EDT in line mode. Here, we activate EDT, enter our job control statements, store our job control stream as a procedure module, and then terminate EDT.

```
Activates EDT                  1.ØØØØ▷// JOB:MYJOB
                               2.ØØØØ▷// DVC 2Ø
                               3.ØØØØ▷// LFD PRNTR
                               4.ØØØØ▷// DVC 5Ø
Job control stream             5.ØØØØ▷// VOL RELØ7Ø
                               6.ØØØØ▷// LBL MYPROGFILE
                               7.ØØØØ▷// LFD SRCLIB
                               8.ØØØØ▷// FOR4LG
                               9.ØØØØ▷/&
Stores our job control        1Ø.ØØØØ▷@WRITE MO=JOB1,FIL=JOBFILE,
stream                                   VSN=RELØ7Ø,TY=P,SAT=YES,SI=4
Terminates EDT                1Ø.ØØØØ▷@HALT
```

G.6. UPDATING A JOB CONTROL STREAM IN LINE MODE

The following example shows you how to update a job control stream using EDT in line mode. Here, we activate EDT and then read in a copy of our job control stream to the workspace file and display it. Next, we make our changes, display the changed statements, and then store the new version of our job control stream (overwriting the previous version). Last, we terminate EDT.

```
Activates EDT                  EDT
Transfers a copy of our file   1.ØØØØ▷@READ MO=JOB1,FIL=JOBFILE,
to the work-space file                  VSN=RELØ7Ø,TY=P
Displays our file             1Ø.ØØØØ▷@PRINT
                               1.ØØØØ▷// JOB MYJOB
                               2.ØØØØ▷// DVC 2Ø
                               3.ØØØØ▷// LFD PRNTR
Our file                       4.ØØØØ▷// DVC 5Ø
                               5.ØØØØ▷// VOL RELØ7Ø
                               6.ØØØØ▷// LBL MYPROGFILE
                               7.ØØØØ▷// LFD SRCLIB
                               8.ØØØØ▷// FOR4LG
                               9.ØØØØ▷/&
Changes 50 to 51 on line 4    1Ø.ØØØØ▷@ON 4 CHANGE '5Ø'TO'51'
Changes SRCLIB to LIBSRC      1Ø.ØØØØ▷@ON 7 CHANGE 'SRCLIB' TO 'LIBSRC'
on line 7
Displays lines 4 and 7        1Ø.ØØØØ▷@PRINT 4,7
                               4.ØØØØ▷// DVC 51
                               7.ØØØØ▷// LFD LIBSRC
```

(continued)

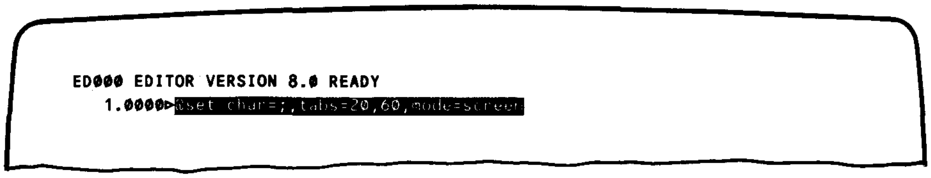
Stores our updated file	`10.0000▷@WRITE MO=JOB1,FIL=JOBFILE,`
	`VSN=REL070,TY=P`

Reminds us that this file already exists, and asks if we want to overwrite it. We select Y, causing the previous version of our file to be overwritten.	`IS100 FILE/MODULE ALREADY EXISTS;`
	`OK TO WRITE TO IT? (Y,N)▷Y`

Terminates EDT	`10.0000▷@HALT`

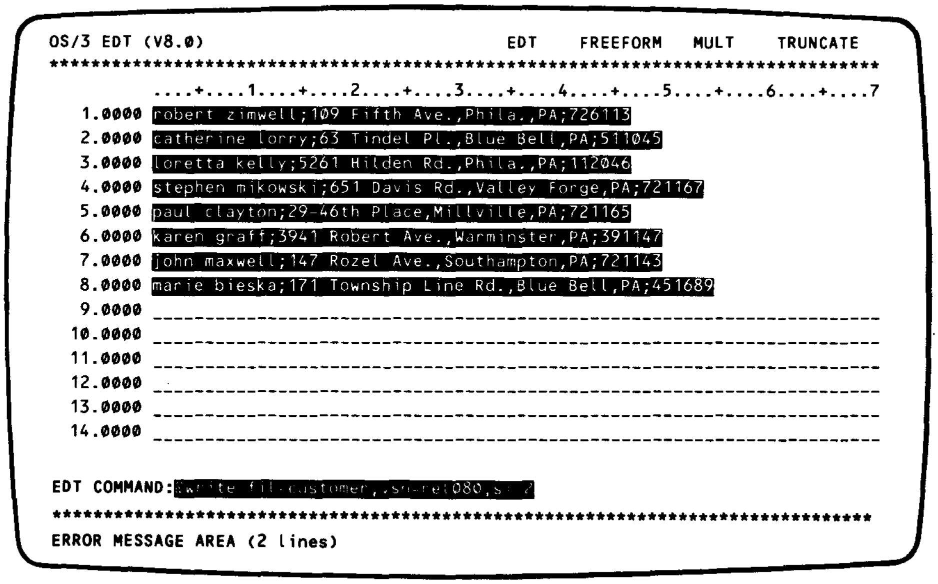
G.7. CREATING A MIRAM DATA FILE IN SCREEN MODE

In this sample screen mode session, we will create a MIRAM data file using the screen-mode freeform screen. The file will contain a list of customer names, addresses, and account numbers. All your entries are shown in white letters on a black background.



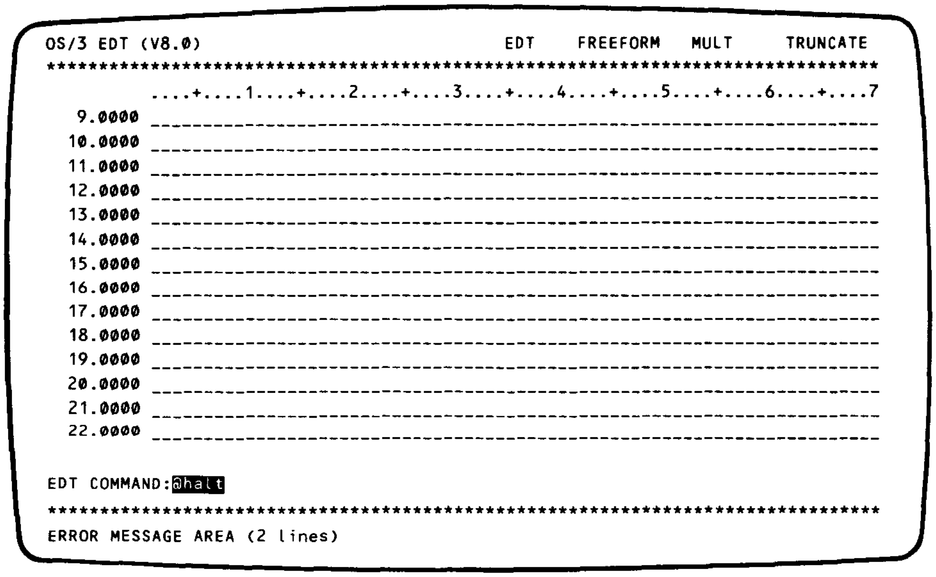
```
ED000 EDITOR VERSION 8.0 READY
   1.0000▷@set char=;,tabs=20,60,mode=screen
```

After you activate EDT (as described in Section 2), EDT displays a message telling you that it's ready and displays the first work space line. You enter an @SET directive to set your tab character to a semicolon and to set your tabs at column 20 and column 60. Setting the tabs lets us format our file into columns containing the names, addresses, and account numbers of our customers. You also include the MODE=SCREEN parameter to enter screen mode. Then you press the XMIT key.



```
OS/3 EDT (V8.0)                              EDT    FREEFORM   MULT    TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  1.0000 robert zimwell;109 Fifth Ave.,Phila.,PA;726113
  2.0000 catherine lorry;63 Tindel Pl.,Blue Bell,PA;511045
  3.0000 loretta kelly;5261 Hilden Rd.,Phila.,PA;112046
  4.0000 stephen mikowski;651 Davis Rd.,Valley Forge,PA;721167
  5.0000 paul clayton;29-46th Place,Millville,PA;721165
  6.0000 karen graff;3941 Robert Ave.,Warminster,PA;391147
  7.0000 john maxwell;147 Rozel Ave.,Southampton,PA;721143
  8.0000 marie bieska;171 Township Line Rd.,Blue Bell,PA;451689
  9.0000 ----------------------------------------------------------------------
 10.0000 ----------------------------------------------------------------------
 11.0000 ----------------------------------------------------------------------
 12.0000 ----------------------------------------------------------------------
 13.0000 ----------------------------------------------------------------------
 14.0000 ----------------------------------------------------------------------

EDT COMMAND:@write fil=customer,vsn=rel080,s=2
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays a screen-mode freeform screen and positions the cursor at the first field on line 1. You enter each name in your data file, press the semicolon key to tell EDT to go to column 20 in its work space, enter each customer's address, press the semicolon key again to go to column 60 in the work space, and enter each customer's account number. After each line, you press the TAB FORWARD key to move the cursor to the next line on the screen. After the eighth data entry, you press the TAB FORWARD key to the EDT COMMAND line, where you enter an @WRITE command to write your MIRAM data file (named CUSTOMER) to the disk volume whose serial number is rel080. You include the SIZE parameter to allocate the file. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.

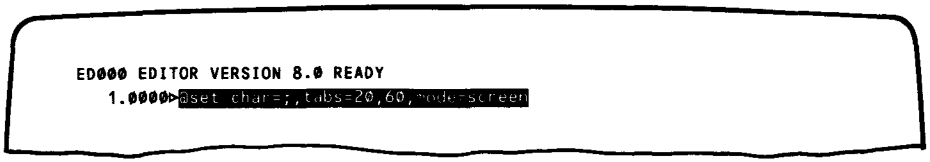


```
OS/3 EDT (V8.0)                              EDT    FREEFORM   MULT    TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  9.0000 ----------------------------------------------------------------------
 10.0000 ----------------------------------------------------------------------
 11.0000 ----------------------------------------------------------------------
 12.0000 ----------------------------------------------------------------------
 13.0000 ----------------------------------------------------------------------
 14.0000 ----------------------------------------------------------------------
 15.0000 ----------------------------------------------------------------------
 16.0000 ----------------------------------------------------------------------
 17.0000 ----------------------------------------------------------------------
 18.0000 ----------------------------------------------------------------------
 19.0000 ----------------------------------------------------------------------
 20.0000 ----------------------------------------------------------------------
 21.0000 ----------------------------------------------------------------------
 22.0000 ----------------------------------------------------------------------

EDT COMMAND:@halt
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another freeform screen and positions the cursor at line 9 since you entered only eight data lines on the previous screen. You press the TAB FORWARD key to the EDT COMMAND line and enter an @HALT directive to end your EDT session. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.

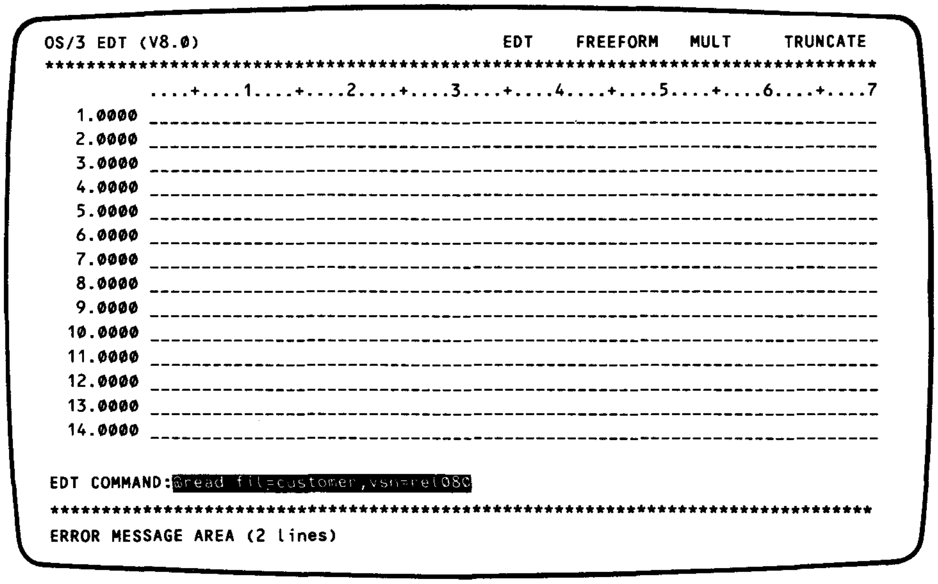
G.8. UPDATING A MIRAM DATA FILE IN SCREEN MODE

The following sample session shows how you update a MIRAM data file in screen mode. The file we'll update is the same file we created in G.7. In this sample session, your entries are shown in white letters on a black background.



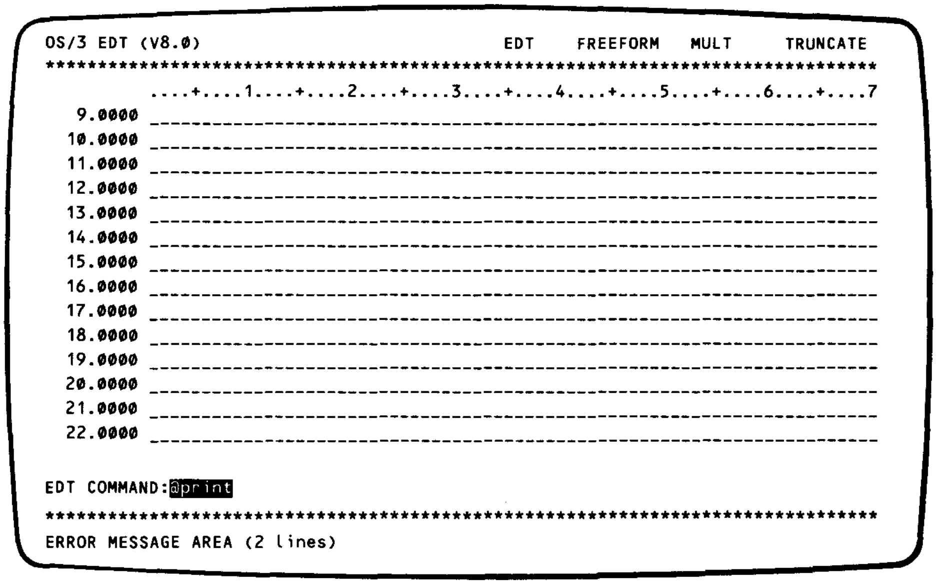
```
ED000 EDITOR VERSION 8.0 READY
   1.0000>@set char=;,tabs=20,60,mode=screen
```

After activating EDT, you see a message telling you EDT is ready. You enter an @SET directive to set your tabs the same way you did when you created the file. You also include the MODE=SCREEN parameter to enter screen mode. Then you press the XMIT key.



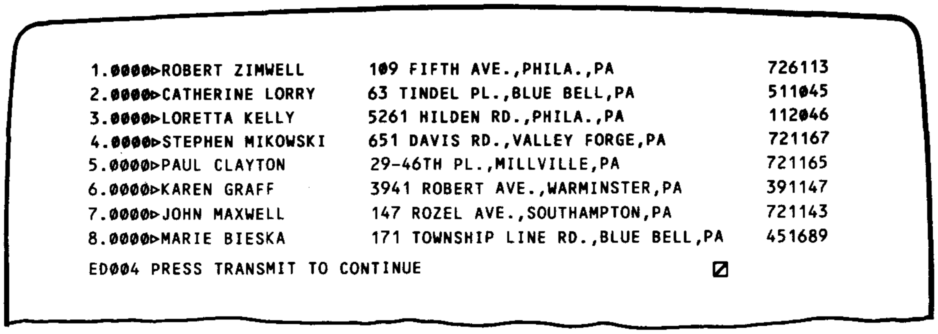
```
OS/3 EDT (V8.0)                                   EDT    FREEFORM   MULT     TRUNCATE
*************************************************************************************
               ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
    1.0000 ______________________________________________________________________
    2.0000 ______________________________________________________________________
    3.0000 ______________________________________________________________________
    4.0000 ______________________________________________________________________
    5.0000 ______________________________________________________________________
    6.0000 ______________________________________________________________________
    7.0000 ______________________________________________________________________
    8.0000 ______________________________________________________________________
    9.0000 ______________________________________________________________________
   10.0000 ______________________________________________________________________
   11.0000 ______________________________________________________________________
   12.0000 ______________________________________________________________________
   13.0000 ______________________________________________________________________
   14.0000 ______________________________________________________________________

EDT COMMAND:@read fil=customer,vsn=rel080
*************************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays a screen-mode freeform screen and positions the cursor at the first field on line 1. You press the TAB FORWARD key to the EDT COMMAND line. Then you enter an @READ command to read the MIRAM data file named CUSTOMER into EDT's work space. This data file resides on the disk volume whose serial number is rel080. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.



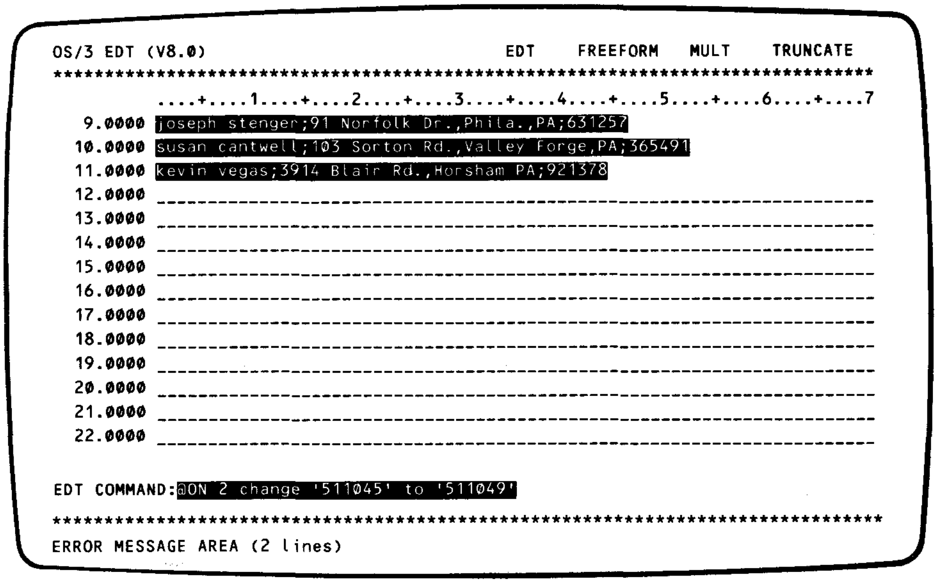
```
OS/3 EDT (V8.Ø)                             EDT    FREEFORM   MULT     TRUNCATE
*******************************************************************************
             ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
    9.ØØØØ ______________________________________________________________________
   1Ø.ØØØØ ______________________________________________________________________
   11.ØØØØ ______________________________________________________________________
   12.ØØØØ ______________________________________________________________________
   13.ØØØØ ______________________________________________________________________
   14.ØØØØ ______________________________________________________________________
   15.ØØØØ ______________________________________________________________________
   16.ØØØØ ______________________________________________________________________
   17.ØØØØ ______________________________________________________________________
   18.ØØØØ ______________________________________________________________________
   19.ØØØØ ______________________________________________________________________
   2Ø.ØØØØ ______________________________________________________________________
   21.ØØØØ ______________________________________________________________________
   22.ØØØØ ______________________________________________________________________

EDT COMMAND:@print
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays the freeform screen with the cursor at line 9 since the file, CUSTOMER, contains eight lines. You press the TAB FORWARD key to the EDT COMMAND line. Then you enter an @PRINT command to display your file so you can see where to change to it. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.



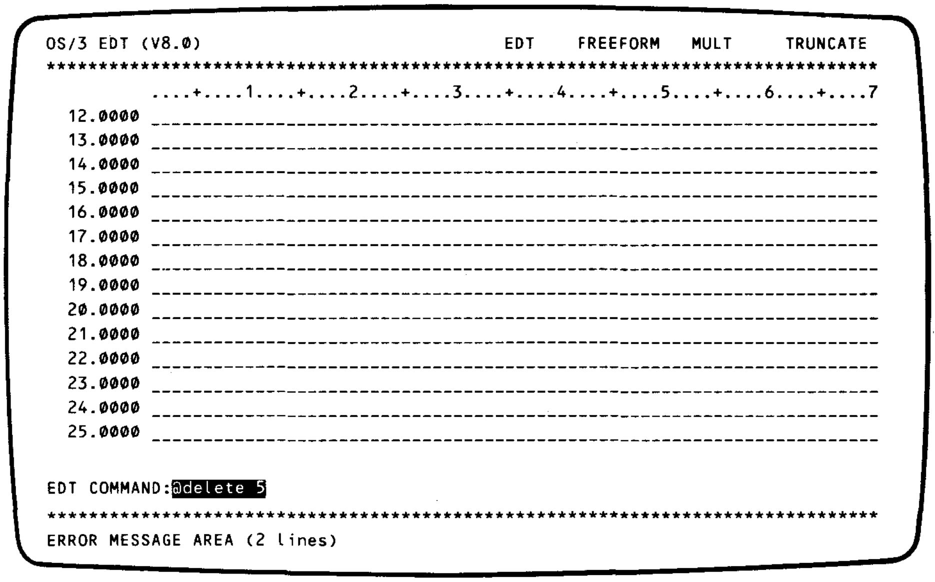
```
1.ØØØØ▷ROBERT ZIMWELL       1Ø9 FIFTH AVE.,PHILA.,PA                726113
2.ØØØØ▷CATHERINE LORRY      63 TINDEL PL.,BLUE BELL,PA              511Ø45
3.ØØØØ▷LORETTA KELLY        5261 HILDEN RD.,PHILA.,PA               112Ø46
4.ØØØØ▷STEPHEN MIKOWSKI     651 DAVIS RD.,VALLEY FORGE,PA           721167
5.ØØØØ▷PAUL CLAYTON         29-46TH PL.,MILLVILLE,PA                721165
6.ØØØØ▷KAREN GRAFF          3941 ROBERT AVE.,WARMINSTER,PA          391147
7.ØØØØ▷JOHN MAXWELL         147 ROZEL AVE.,SOUTHAMPTON,PA           721143
8.ØØØØ▷MARIE BIESKA         171 TOWNSHIP LINE RD.,BLUE BELL,PA      451689
EDØØ4 PRESS TRANSMIT TO CONTINUE                                ⧄
```

EDT displays the file with your customers' names starting in column 1, their addresses starting in column 20, and their account numbers in column 60. EDT formats its display this way because you set and used those columns as tabs when you originally created the file in G.7. You note the lines you want to change and what changes you need to make. Then you press the XMIT key.



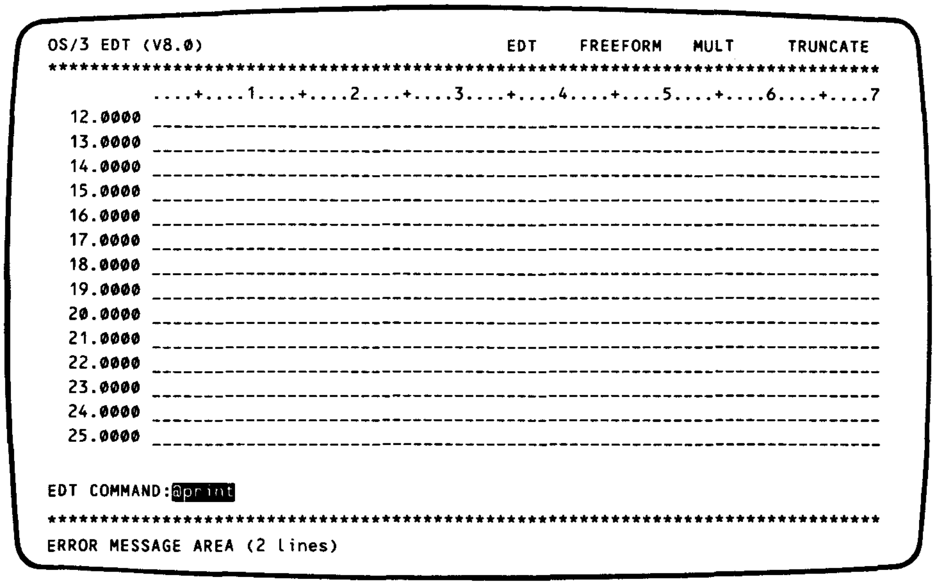
```
OS/3 EDT (V8.Ø)                                EDT     FREEFORM   MULT     TRUNCATE
********************************************************************************
           ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
   9.ØØØØ joseph stenger;91 Norfolk Dr.,Phila.,PA;631257
  1Ø.ØØØØ susan cantwell;1Ø3 Sorton Rd.,Valley Forge,PA;365491
  11.ØØØØ kevin vegas;3914 Blair Rd.,Horsham PA;921378
  12.ØØØØ ______________________________________________________________________
  13.ØØØØ ______________________________________________________________________
  14.ØØØØ ______________________________________________________________________
  15.ØØØØ ______________________________________________________________________
  16.ØØØØ ______________________________________________________________________
  17.ØØØØ ______________________________________________________________________
  18.ØØØØ ______________________________________________________________________
  19.ØØØØ ______________________________________________________________________
  2Ø.ØØØØ ______________________________________________________________________
  21.ØØØØ ______________________________________________________________________
  22.ØØØØ ______________________________________________________________________

EDT COMMAND:@ON 2 change '511Ø45' to '511Ø49'
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays the freeform screen again and positions the cursor at line 9. Now you are ready to make changes. You enter three more records to CUSTOMER, pressing the TAB FORWARD key after each line. After keying in the third new record, you press the TAB FORWARD key to the EDT COMMAND line where you enter an @CHANGE command to change Catherine Lorry's account number from 511045 to 511049. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key to pass all of the information on this screen to EDT.



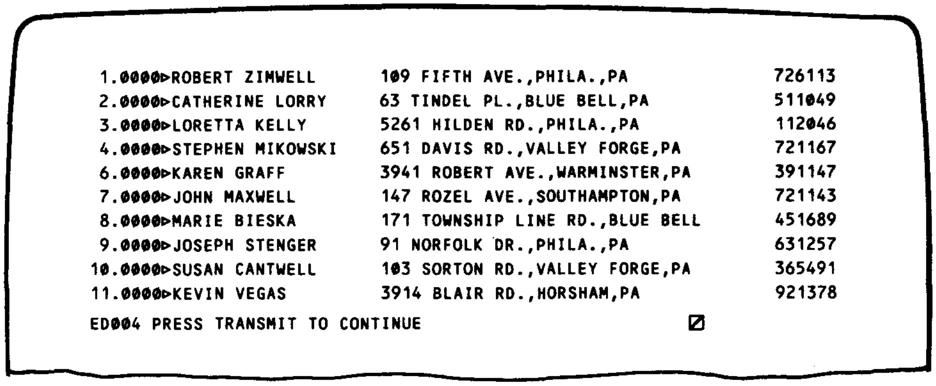
```
OS/3 EDT (V8.Ø)                                EDT     FREEFORM   MULT     TRUNCATE
********************************************************************************
           ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  12.ØØØØ ______________________________________________________________________
  13.ØØØØ ______________________________________________________________________
  14.ØØØØ ______________________________________________________________________
  15.ØØØØ ______________________________________________________________________
  16.ØØØØ ______________________________________________________________________
  17.ØØØØ ______________________________________________________________________
  18.ØØØØ ______________________________________________________________________
  19.ØØØØ ______________________________________________________________________
  2Ø.ØØØØ ______________________________________________________________________
  21.ØØØØ ______________________________________________________________________
  22.ØØØØ ______________________________________________________________________
  23.ØØØØ ______________________________________________________________________
  24.ØØØØ ______________________________________________________________________
  25.ØØØØ ______________________________________________________________________

EDT COMMAND:@delete 5
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another freeform screen and positions the cursor at line 12 since CUSTOMER now contains 11 lines. You press the TAB FORWARD key to the EDT COMMAND line where you enter an @DELETE command to delete Paul Clayton's records (on line 5) from the file. Then you press the TAB FORWARD key to the bottom rightmost positon on the screen and press the XMIT key.



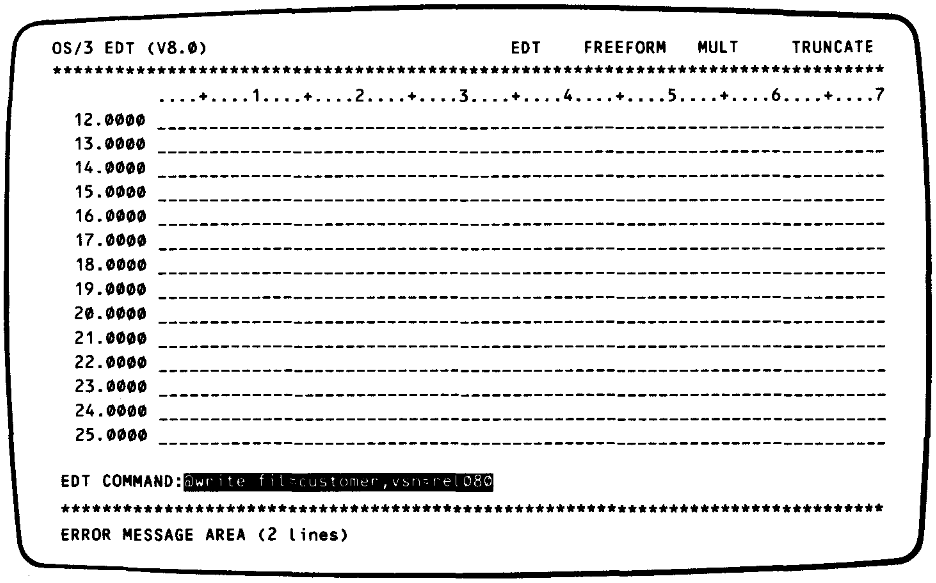
```
OS/3 EDT (V8.Ø)                              EDT     FREEFORM    MULT      TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
 12.ØØØØ  ______________________________________________________________________
 13.ØØØØ  ______________________________________________________________________
 14.ØØØØ  ______________________________________________________________________
 15.ØØØØ  ______________________________________________________________________
 16.ØØØØ  ______________________________________________________________________
 17.ØØØØ  ______________________________________________________________________
 18.ØØØØ  ______________________________________________________________________
 19.ØØØØ  ______________________________________________________________________
 2Ø.ØØØØ  ______________________________________________________________________
 21.ØØØØ  ______________________________________________________________________
 22.ØØØØ  ______________________________________________________________________
 23.ØØØØ  ______________________________________________________________________
 24.ØØØØ  ______________________________________________________________________
 25.ØØØØ  ______________________________________________________________________

EDT COMMAND:@print
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another freeform screen. You press the TAB FORWARD key again to the EDT COMMAND line where you enter an @PRINT command so you can see the changes. Then you press the TAB FORWARD key to the bottom rightmost positon on the screen and press the XMIT key.



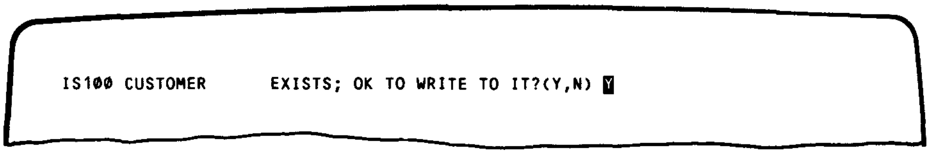
```
  1.ØØØØ▷ROBERT ZIMWELL      1Ø9 FIFTH AVE.,PHILA.,PA                726113
  2.ØØØØ▷CATHERINE LORRY     63 TINDEL PL.,BLUE BELL,PA              511Ø49
  3.ØØØØ▷LORETTA KELLY       5261 HILDEN RD.,PHILA.,PA               112Ø46
  4.ØØØØ▷STEPHEN MIKOWSKI    651 DAVIS RD.,VALLEY FORGE,PA           721167
  6.ØØØØ▷KAREN GRAFF         3941 ROBERT AVE.,WARMINSTER,PA          391147
  7.ØØØØ▷JOHN MAXWELL        147 ROZEL AVE.,SOUTHAMPTON,PA           721143
  8.ØØØØ▷MARIE BIESKA        171 TOWNSHIP LINE RD.,BLUE BELL         451689
  9.ØØØØ▷JOSEPH STENGER      91 NORFOLK DR.,PHILA.,PA                631257
 1Ø.ØØØØ▷SUSAN CANTWELL      1Ø3 SORTON RD.,VALLEY FORGE,PA          365491
 11.ØØØØ▷KEVIN VEGAS         3914 BLAIR RD.,HORSHAM,PA               921378
EDØØ4 PRESS TRANSMIT TO CONTINUE                             ☒
```

EDT displays the updated version of your file. You see the three new records, the change in Catherine Lorry's account number, and the absence of Paul Clayton's records from the file. Then you press the XMIT key to continue.



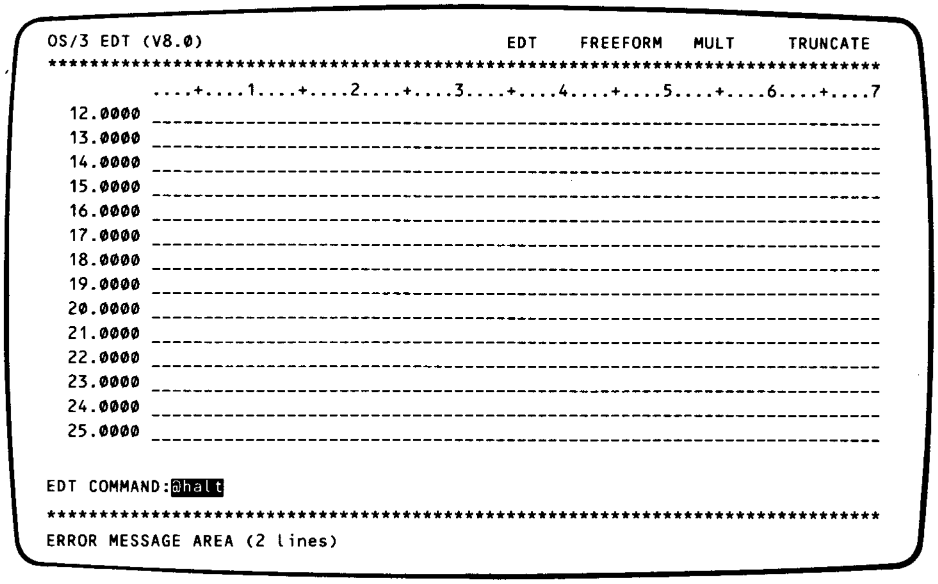
```
OS/3 EDT (V8.Ø)                                EDT   FREEFORM   MULT     TRUNCATE
*********************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
 12.ØØØØ  ________________________________________________________________________
 13.ØØØØ  ________________________________________________________________________
 14.ØØØØ  ________________________________________________________________________
 15.ØØØØ  ________________________________________________________________________
 16.ØØØØ  ________________________________________________________________________
 17.ØØØØ  ________________________________________________________________________
 18.ØØØØ  ________________________________________________________________________
 19.ØØØØ  ________________________________________________________________________
 2Ø.ØØØØ  ________________________________________________________________________
 21.ØØØØ  ________________________________________________________________________
 22.ØØØØ  ________________________________________________________________________
 23.ØØØØ  ________________________________________________________________________
 24.ØØØØ  ________________________________________________________________________
 25.ØØØØ  ________________________________________________________________________

EDT COMMAND:@write fil=customer,vsn=rel080
*********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another freeform screen. You press the TAB FORWARD key to the EDT COMMAND line where you enter an @WRITE command to write your file back to the disk volume. Then you press the TAB FORWARD KEY to the bottom rightmost position on the screen and press the XMIT key.



```
IS1ØØ CUSTOMER      EXISTS; OK TO WRITE TO IT?(Y,N) Y
```

EDT reminds you that your file already exists and double-checks if you want to overwrite it. You want to overwrite it, so you respond Y for yes and press the XMIT key.



```
OS/3 EDT (V8.0)                                      EDT    FREEFORM   MULT     TRUNCATE
******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
 12.0000  ______________________________________________________________________
 13.0000  ______________________________________________________________________
 14.0000  ______________________________________________________________________
 15.0000  ______________________________________________________________________
 16.0000  ______________________________________________________________________
 17.0000  ______________________________________________________________________
 18.0000  ______________________________________________________________________
 19.0000  ______________________________________________________________________
 20.0000  ______________________________________________________________________
 21.0000  ______________________________________________________________________
 22.0000  ______________________________________________________________________
 23.0000  ______________________________________________________________________
 24.0000  ______________________________________________________________________
 25.0000  ______________________________________________________________________

EDT COMMAND:@halt
******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

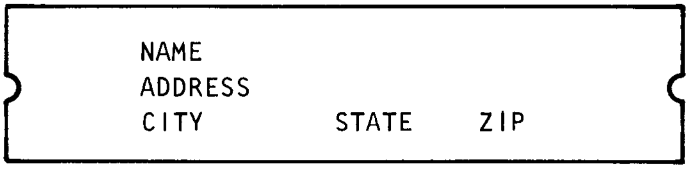
EDT displays another freeform screen and positions the cursor at line 12. You press TAB FORWARD key to the EDT COMMAND line and enter an @HALT directive to end your EDT session. Then you press the TAB FORWARD key again to the bottom rightmost position on the screen and press the XMIT key. EDT terminates.

G.9. UPDATING A USER PROGRAM (SOURCE MODULE) IN SCREEN MODE

The following sample session shows how you update a COBOL source program in screen mode. In this session, we'll update the same program we updated in Section 2. And we'll make the same changes to it. The difference is that this is a screen mode session and the session in Section 2 is a line mode session.

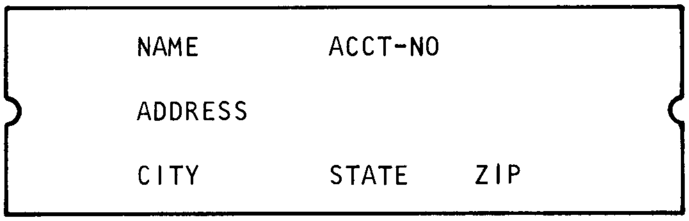
Because we recommend you use screen mode for updating, not creating, either COBOL or RPG II source programs, here we'll show only an updating session. Keep in mind that you can create a COBOL source program either in EDT line mode or, better yet, using the COBOL editor, COBEDT.

If you remember, our program LABELS prints address labels for magazines, like this:



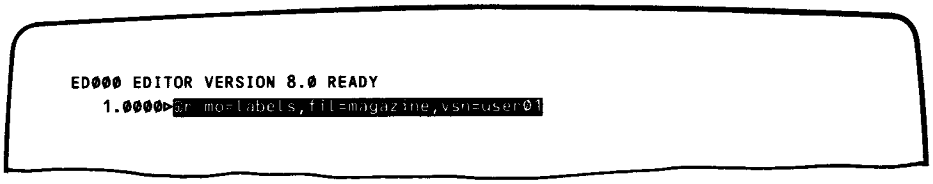
```
NAME
ADDRESS
CITY          STATE      ZIP
```

We're going to update LABELS to double-space the lines and include an account number next to the name, like this:



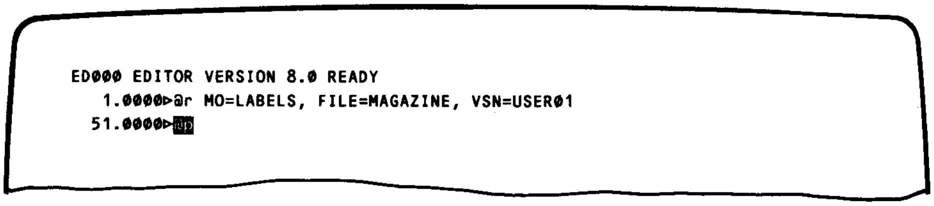
```
NAME         ACCT-NO

ADDRESS

CITY         STATE    ZIP
```

In this sample session, your entries are shown in white letters on a black background.



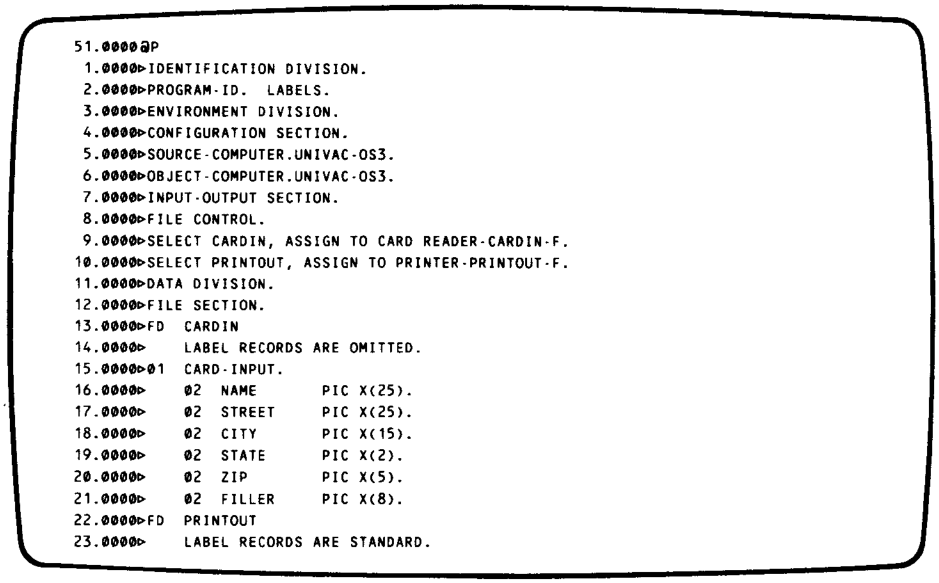
```
ED000 EDITOR VERSION 8.0 READY
   1.0000▷@r mo=labels,fil=magazine,vsn=user01
```

After activating EDT, you see a message telling you EDT is ready. You then enter an @READ command to read your source module LABELS to the EDT work space from the file named MAGAZINE on the volume whose serial number is USER01. Then you press the XMIT key.



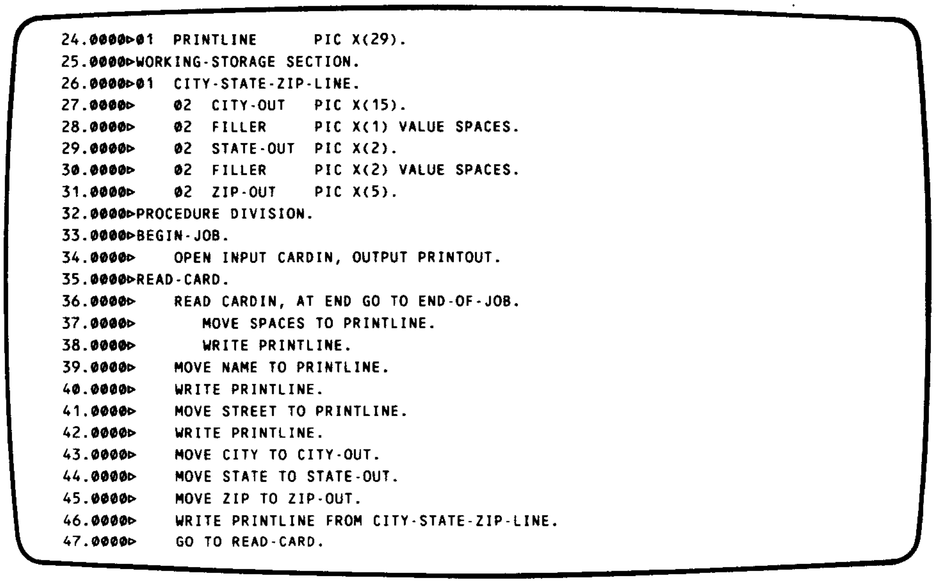
```
ED000 EDITOR VERSION 8.0 READY
   1.0000▷@r MO=LABELS, FILE=MAGAZINE, VSN=USER01
  51.0000▷@p
```

EDT copies the module into its work space and positions the cursor at line 51 since LABELS contains 50 lines of source code. You enter an @PRINT command to display LABELS on your workstation screen and press the XMIT key.



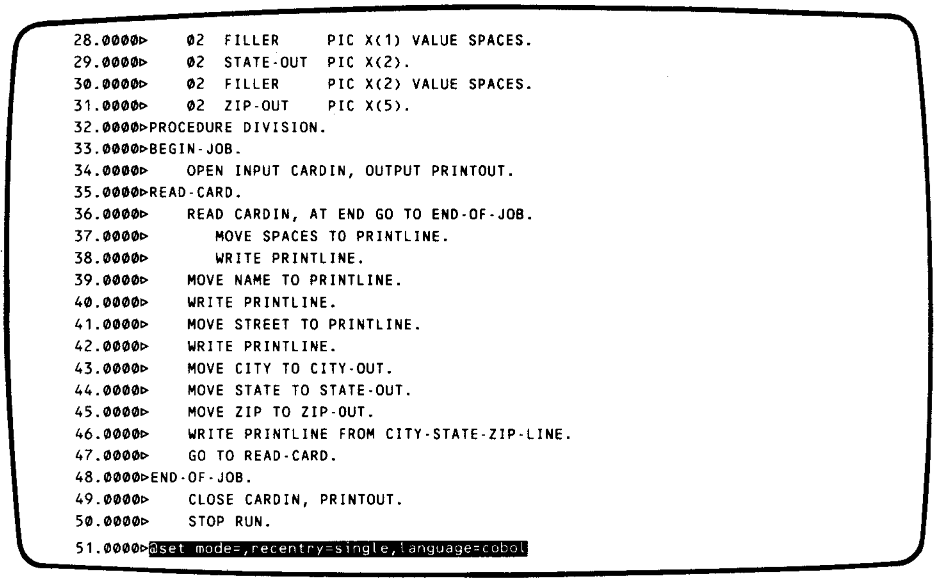
```
51.0000@P
 1.0000▷IDENTIFICATION DIVISION.
 2.0000▷PROGRAM-ID.  LABELS.
 3.0000▷ENVIRONMENT DIVISION.
 4.0000▷CONFIGURATION SECTION.
 5.0000▷SOURCE-COMPUTER.UNIVAC-OS3.
 6.0000▷OBJECT-COMPUTER.UNIVAC-OS3.
 7.0000▷INPUT-OUTPUT SECTION.
 8.0000▷FILE CONTROL.
 9.0000▷SELECT CARDIN, ASSIGN TO CARD READER-CARDIN-F.
10.0000▷SELECT PRINTOUT, ASSIGN TO PRINTER-PRINTOUT-F.
11.0000▷DATA DIVISION.
12.0000▷FILE SECTION.
13.0000▷FD  CARDIN
14.0000▷    LABEL RECORDS ARE OMITTED.
15.0000▷01  CARD-INPUT.
16.0000▷    02  NAME       PIC X(25).
17.0000▷    02  STREET     PIC X(25).
18.0000▷    02  CITY       PIC X(15).
19.0000▷    02  STATE      PIC X(2).
20.0000▷    02  ZIP        PIC X(5).
21.0000▷    02  FILLER     PIC X(8).
22.0000▷FD  PRINTOUT
23.0000▷    LABEL RECORDS ARE STANDARD.
```

EDT displays the first 23 lines of LABELS on your workstation screen. You note the lines that require updating and press the F19 function key, while pressing the FUNCTION key, to see succeeding source code lines.



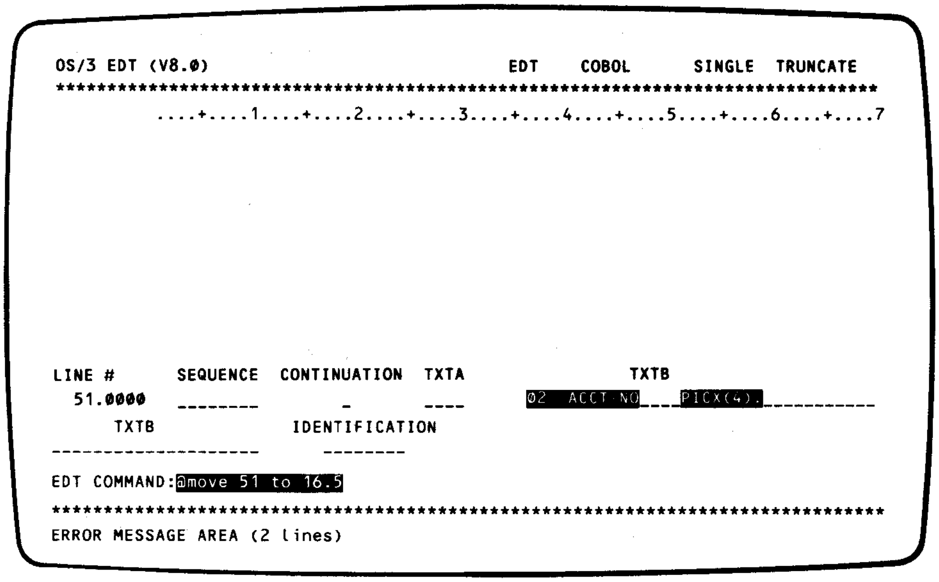
```
24.0000▷01  PRINTLINE      PIC X(29).
25.0000▷WORKING-STORAGE SECTION.
26.0000▷01  CITY-STATE-ZIP-LINE.
27.0000▷    02  CITY-OUT   PIC X(15).
28.0000▷    02  FILLER     PIC X(1) VALUE SPACES.
29.0000▷    02  STATE-OUT  PIC X(2).
30.0000▷    02  FILLER     PIC X(2) VALUE SPACES.
31.0000▷    02  ZIP-OUT    PIC X(5).
32.0000▷PROCEDURE DIVISION.
33.0000▷BEGIN-JOB.
34.0000▷    OPEN INPUT CARDIN, OUTPUT PRINTOUT.
35.0000▷READ-CARD.
36.0000▷    READ CARDIN, AT END GO TO END-OF-JOB.
37.0000▷       MOVE SPACES TO PRINTLINE.
38.0000▷       WRITE PRINTLINE.
39.0000▷    MOVE NAME TO PRINTLINE.
40.0000▷    WRITE PRINTLINE.
41.0000▷    MOVE STREET TO PRINTLINE.
42.0000▷    WRITE PRINTLINE.
43.0000▷    MOVE CITY TO CITY-OUT.
44.0000▷    MOVE STATE TO STATE-OUT.
45.0000▷    MOVE ZIP TO ZIP-OUT.
46.0000▷    WRITE PRINTLINE FROM CITY-STATE-ZIP-LINE.
47.0000▷    GO TO READ-CARD.
```

EDT displays lines 24 through 47 of LABELS. Again, you note where to make changes and press the F19 function key to see the last three lines of LABELS.



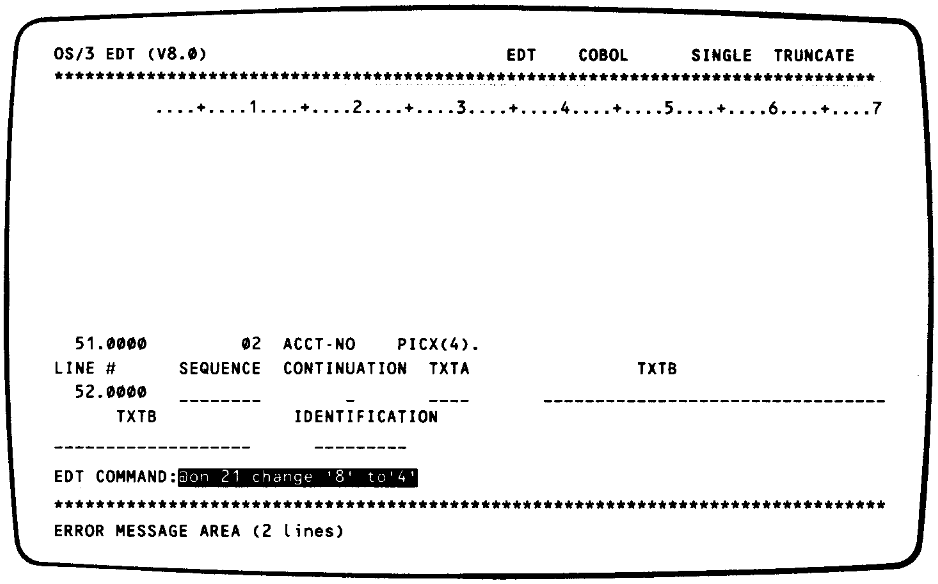
```
28.ØØØØ▷    Ø2  FILLER     PIC X(1) VALUE SPACES.
29.ØØØØ▷    Ø2  STATE-OUT  PIC X(2).
3Ø.ØØØØ▷    Ø2  FILLER     PIC X(2) VALUE SPACES.
31.ØØØØ▷    Ø2  ZIP-OUT    PIC X(5).
32.ØØØØ▷PROCEDURE DIVISION.
33.ØØØØ▷BEGIN-JOB.
34.ØØØØ▷    OPEN INPUT CARDIN, OUTPUT PRINTOUT.
35.ØØØØ▷READ-CARD.
36.ØØØØ▷    READ CARDIN, AT END GO TO END-OF-JOB.
37.ØØØØ▷       MOVE SPACES TO PRINTLINE.
38.ØØØØ▷       WRITE PRINTLINE.
39.ØØØØ▷    MOVE NAME TO PRINTLINE.
4Ø.ØØØØ▷    WRITE PRINTLINE.
41.ØØØØ▷    MOVE STREET TO PRINTLINE.
42.ØØØØ▷    WRITE PRINTLINE.
43.ØØØØ▷    MOVE CITY TO CITY-OUT.
44.ØØØØ▷    MOVE STATE TO STATE-OUT.
45.ØØØØ▷    MOVE ZIP TO ZIP-OUT.
46.ØØØØ▷    WRITE PRINTLINE FROM CITY-STATE-ZIP-LINE.
47.ØØØØ▷    GO TO READ-CARD.
48.ØØØØ▷END-OF-JOB.
49.ØØØØ▷    CLOSE CARDIN, PRINTOUT.
5Ø.ØØØØ▷    STOP RUN.
51.ØØØØ▷@set mode=,recentry=single,language=cobol
```

EDT scrolls the lines up and displays the rest of LABELS. It positions the cursor at line 51 to accept an EDT command. You check if the last three lines of LABELS require changes, then enter an @SET directive to enter screen mode. You include the RECENTRY=SINGLE parameter because you'll be updating one line at a time. You also include the LANGUAGE=COBOL parameter to use a COBOL screen. Then you press the XMIT key.



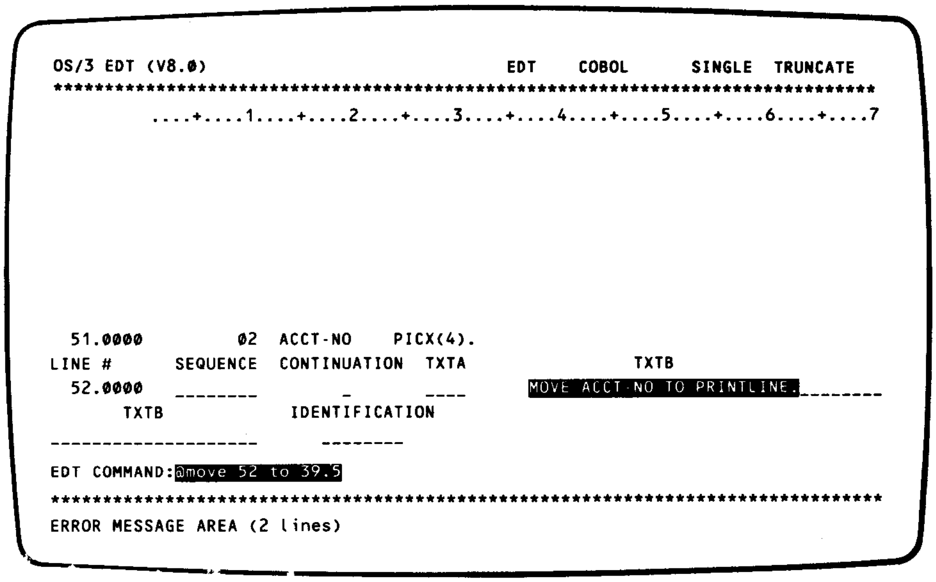
```
OS/3 EDT (V8.Ø)                                  EDT     COBOL      SINGLE  TRUNCATE
*************************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7









LINE #       SEQUENCE  CONTINUATION  TXTA             TXTB
 51.ØØØØ     ________        _       ____     Ø2  ACCT-NO____PICX(4).____________
      TXTB              IDENTIFICATION
---------------------     --------
EDT COMMAND:@move 51 to 16.5
*************************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays the COBOL screen mode screen. It positions the cursor at line 51, ready to accept an entry for the SEQUENCE field. You press the TAB FORWARD key to the TXTB field and key in your new source line, making sure that the line is spaced properly. Then you press the TAB FORWARD key to the EDT COMMAND line where you enter an @MOVE command to move the line you just keyed in to line 16.5. After tabbing to the bottom rightmost position on the screen, you press the XMIT key.



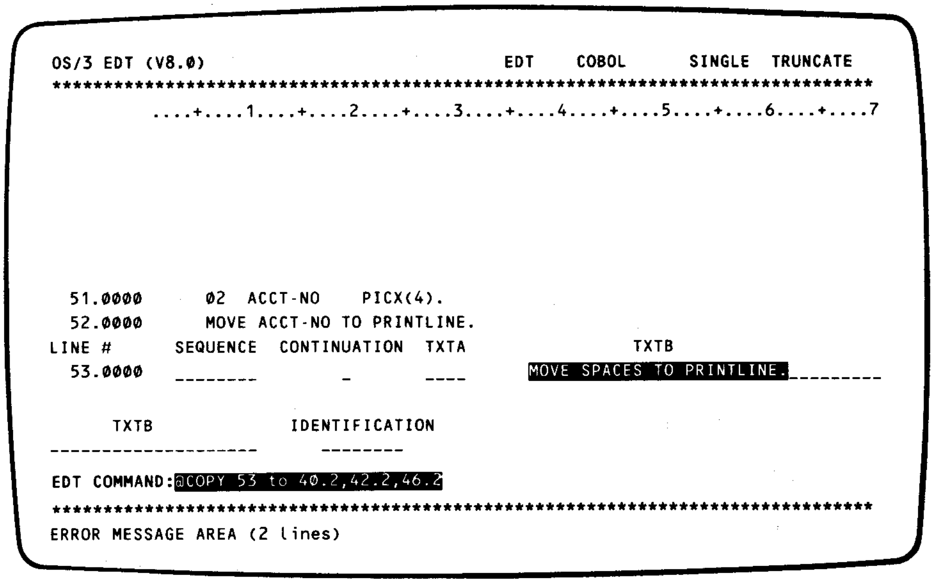
```
OS/3 EDT (V8.0)                                  EDT    COBOL     SINGLE  TRUNCATE
*************************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7




 51.0000         02  ACCT-NO    PICX(4).
LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
 52.0000    --------       -        ----       --------------------------------
    TXTB               IDENTIFICATION
--------------------    ---------
EDT COMMAND:@on 21 change '8' to'4'
*************************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays the same COBOL screen again, except that it now shows the new source line you just added. Because LABELS now contains 51 source lines, it positions the cursor at line 52. You tab to the EDT COMMAND line and enter an @CHANGE command to change the 8 on line 21 to a 4. Then you press the TAB FORWARD key to the bottom rightmost position of the screen and press the XMIT key.



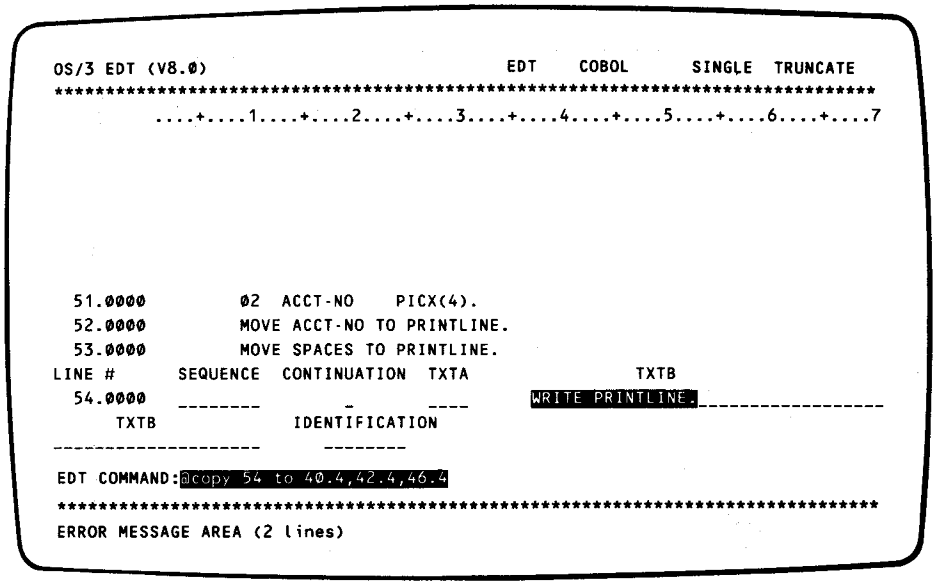
```
OS/3 EDT (V8.0)                                  EDT    COBOL     SINGLE  TRUNCATE
*************************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7




 51.0000         02  ACCT-NO    PICX(4).
LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
 52.0000    --------       -        ----      MOVE ACCT-NO TO PRINTLINE.--------
    TXTB               IDENTIFICATION
--------------------    ---------
EDT COMMAND:@move 52 to 39.5
*************************************************************************************
ERROR MESSAGE AREA (2 lines)
```

On this COBOL screen, you enter a new source line in the TXTB field. Then you press the TAB FORWARD key to the EDT COMMAND line where you enter an @MOVE command to move the new line to line 39.5. After pressing the TAB FORWARD key to the bottom rightmost position on the screen, you press the XMIT key.



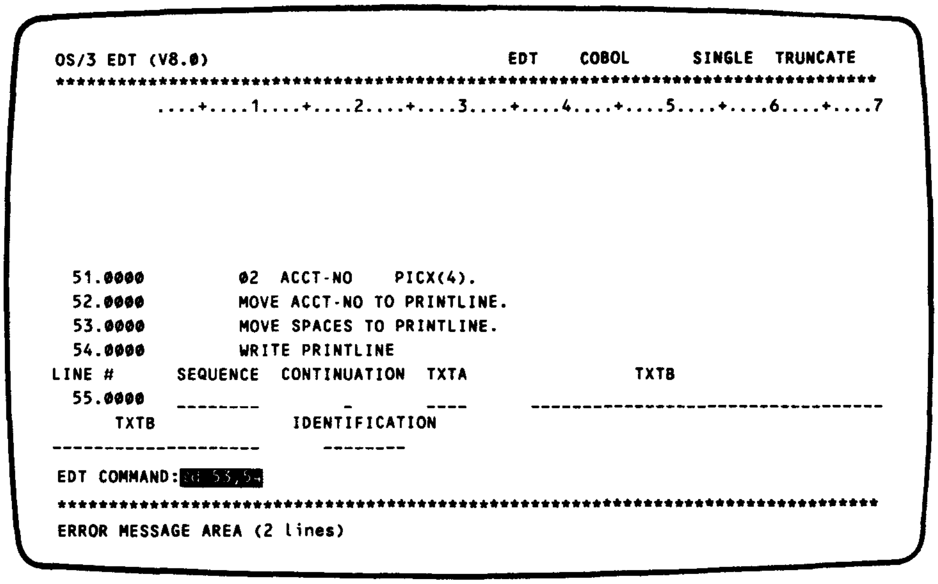
```
OS/3 EDT (V8.Ø)                          EDT    COBOL      SINGLE  TRUNCATE
*******************************************************************************
             ....+....1....+....2....+....3....+....4....+....5....+....6....+....7








 51.ØØØØ       Ø2  ACCT-NO    PICX(4).
 52.ØØØØ       MOVE ACCT-NO TO PRINTLINE.
LINE #      SEQUENCE  CONTINUATION  TXTA                 TXTB
 53.ØØØØ    ________        _       ____        MOVE SPACES TO PRINTLINE._________

     TXTB                 IDENTIFICATION
--------------------         --------
EDT COMMAND:@COPY 53 to 40.2,42.2,46.2
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

On this screen, you add a new source line starting in the TXTB field. Then you enter an @COPY command to copy it to three different places in your program: lines 40.2, 42.2, and 46.2. Then you press the XMIT key.



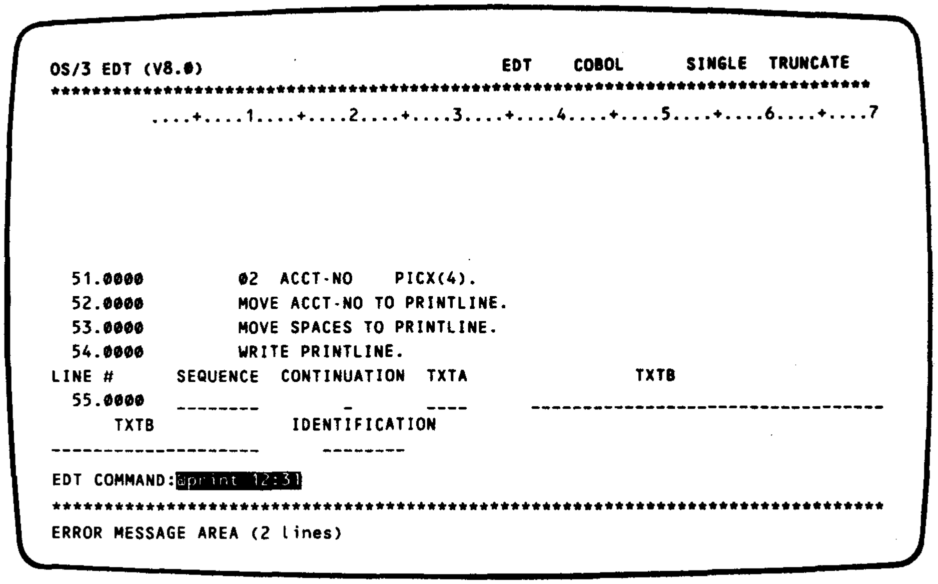
```
OS/3 EDT (V8.Ø)                          EDT    COBOL      SINGLE  TRUNCATE
*******************************************************************************
             ....+....1....+....2....+....3....+....4....+....5....+....6....+....7








 51.ØØØØ          Ø2  ACCT-NO    PICX(4).
 52.ØØØØ          MOVE ACCT-NO TO PRINTLINE.
 53.ØØØØ          MOVE SPACES TO PRINTLINE.
LINE #      SEQUENCE  CONTINUATION  TXTA                 TXTB
 54.ØØØØ    ________        _       ____        WRITE PRINTLINE._________________
     TXTB                 IDENTIFICATION
--------------------         --------
EDT COMMAND:@copy 54 to 40.4,42.4,46.4
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

You key in another source line and copy this one also to three places in your program. Then you press the XMIT key.



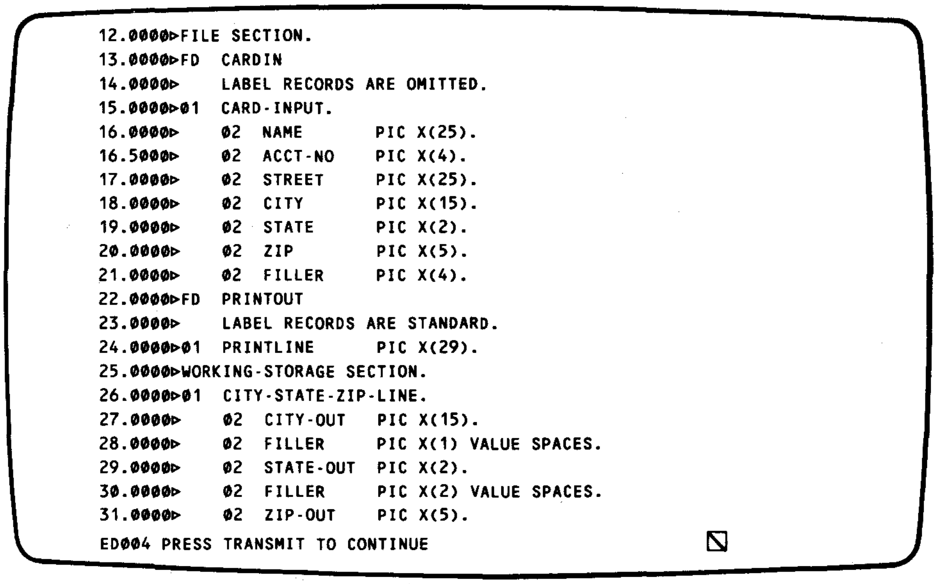
```
OS/3 EDT (V8.Ø)                                   EDT    COBOL      SINGLE  TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7




 51.ØØØØ        Ø2  ACCT-NO    PICX(4).
 52.ØØØØ        MOVE ACCT-NO TO PRINTLINE.
 53.ØØØØ        MOVE SPACES TO PRINTLINE.
 54.ØØØØ        WRITE PRINTLINE
LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
 55.ØØØØ    --------        -       ----      ---------------------------------
    TXTB              IDENTIFICATION
--------------------      --------
EDT COMMAND:@d 53,54
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

You have no more source lines to add, so you press the TAB FORWARD key to the EDT COMMAND line. Because you've already copied lines 53 and 54 to their correct lines, the last change you must make is to delete lines 53 and 54 from EDT's work space. So, you enter the @DELETE command. Then you press the XMIT key.



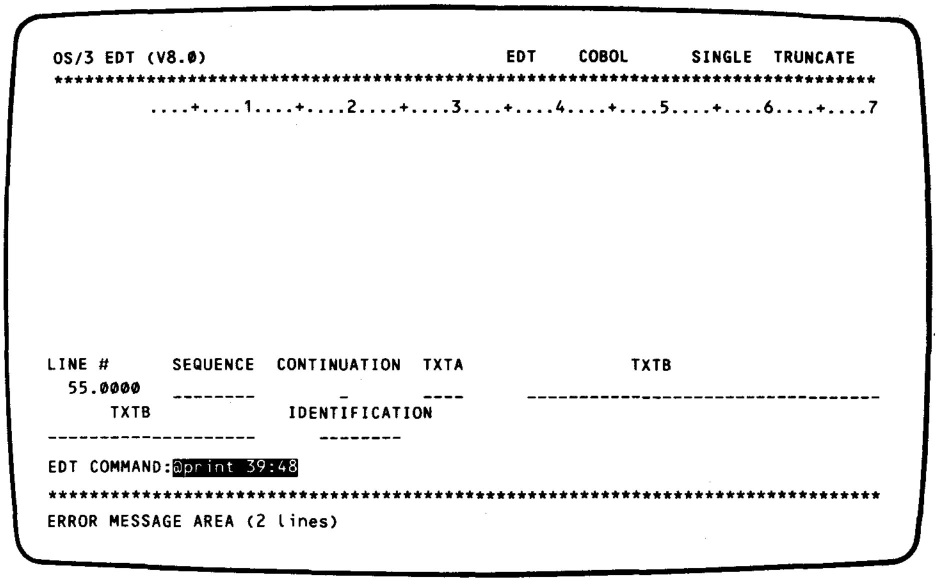
```
OS/3 EDT (V8.Ø)                                   EDT    COBOL      SINGLE  TRUNCATE
*******************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7




 51.ØØØØ        Ø2  ACCT-NO    PICX(4).
 52.ØØØØ        MOVE ACCT-NO TO PRINTLINE.
 53.ØØØØ        MOVE SPACES TO PRINTLINE.
 54.ØØØØ        WRITE PRINTLINE.
LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
 55.ØØØØ    --------        -       ----      ---------------------------------
    TXTB              IDENTIFICATION
--------------------      --------
EDT COMMAND:@print 12:31
*******************************************************************************
ERROR MESSAGE AREA (2 lines)
```

Now that you've made all your changes to your program, you issue an @PRINT command to make sure you made the changes correctly. First you tell EDT to display lines 12 through 31 by issuing @PRINT 12:31. Then you press the XMIT key.



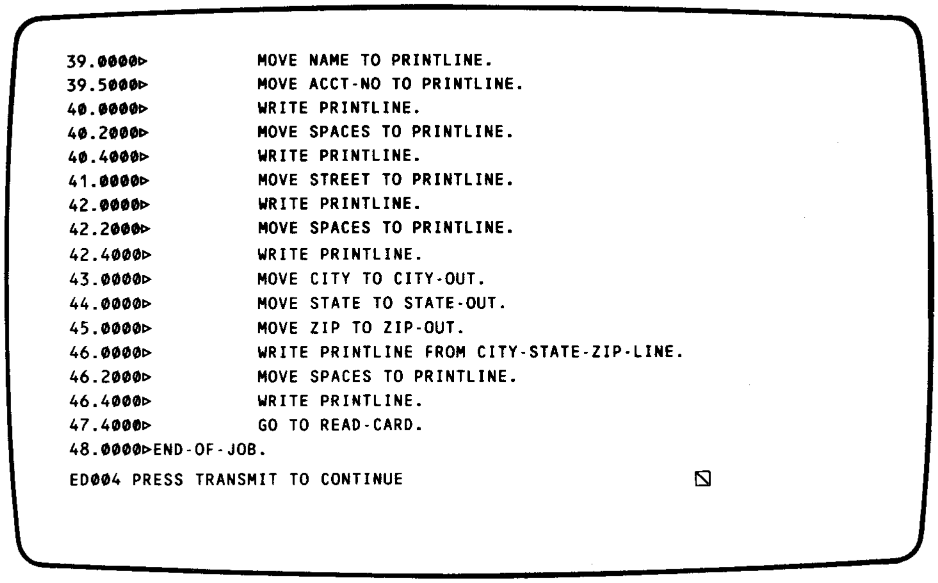
```
12.ØØØØ▷FILE SECTION.
13.ØØØØ▷FD  CARDIN
14.ØØØØ▷    LABEL RECORDS ARE OMITTED.
15.ØØØØ▷Ø1  CARD-INPUT.
16.ØØØØ▷    Ø2  NAME        PIC X(25).
16.5ØØØ▷    Ø2  ACCT-NO     PIC X(4).
17.ØØØØ▷    Ø2  STREET      PIC X(25).
18.ØØØØ▷    Ø2  CITY        PIC X(15).
19.ØØØØ▷    Ø2  STATE       PIC X(2).
2Ø.ØØØØ▷    Ø2  ZIP         PIC X(5).
21.ØØØØ▷    Ø2  FILLER      PIC X(4).
22.ØØØØ▷FD  PRINTOUT
23.ØØØØ▷    LABEL RECORDS ARE STANDARD.
24.ØØØØ▷Ø1  PRINTLINE       PIC X(29).
25.ØØØØ▷WORKING-STORAGE SECTION.
26.ØØØØ▷Ø1  CITY-STATE-ZIP-LINE.
27.ØØØØ▷    Ø2  CITY-OUT    PIC X(15).
28.ØØØØ▷    Ø2  FILLER      PIC X(1) VALUE SPACES.
29.ØØØØ▷    Ø2  STATE-OUT   PIC X(2).
3Ø.ØØØØ▷    Ø2  FILLER      PIC X(2) VALUE SPACES.
31.ØØØØ▷    Ø2  ZIP-OUT     PIC X(5).
EDØØ4 PRESS TRANSMIT TO CONTINUE                          ⧄
```

EDT displays the lines so you can see the new line added on line 16.5 and the change on line 21. You then press the XMIT key to continue.



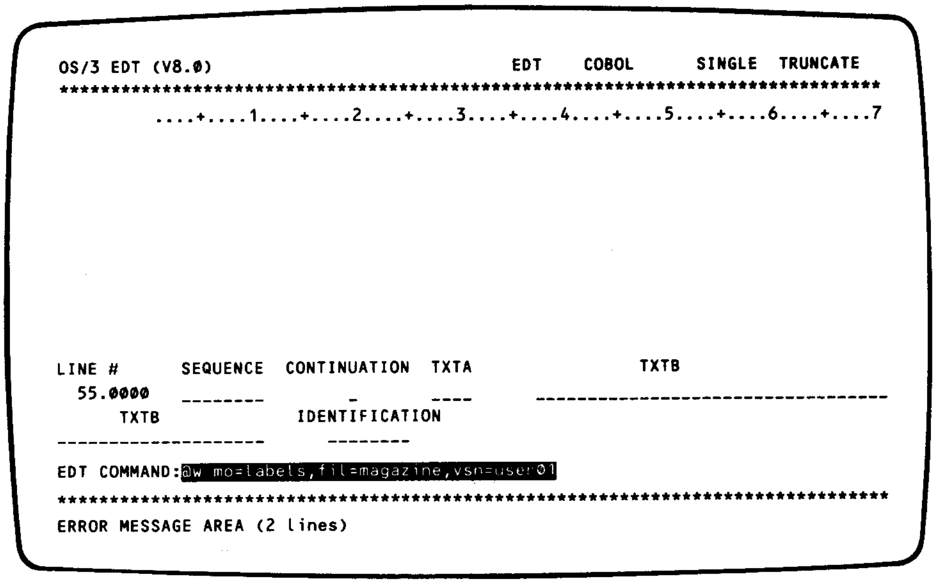
```
OS/3 EDT (V8.Ø)                             EDT    COBOL     SINGLE  TRUNCATE
*********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7








LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
  55.ØØØØ   ________        _       ____      __________________________________
     TXTB              IDENTIFICATION
____________________    ________
EDT COMMAND:@print 39:48
*********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another COBOL screen where you press the TAB FORWARD key to the EDT COMMAND line. Once the cursor is there, you enter a @PRINT command to see the other lines you made changes to. Then you press the XMIT key.



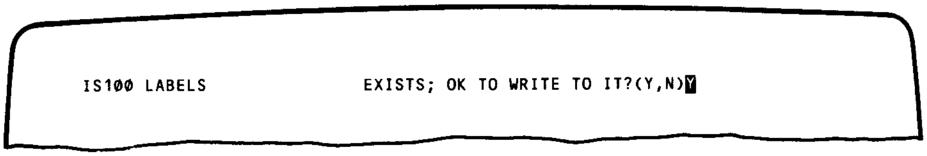
```
39.ØØØØ▷          MOVE NAME TO PRINTLINE.
39.5ØØØ▷          MOVE ACCT-NO TO PRINTLINE.
4Ø.ØØØØ▷          WRITE PRINTLINE.
4Ø.2ØØØ▷          MOVE SPACES TO PRINTLINE.
4Ø.4ØØØ▷          WRITE PRINTLINE.
41.ØØØØ▷          MOVE STREET TO PRINTLINE.
42.ØØØØ▷          WRITE PRINTLINE.
42.2ØØØ▷          MOVE SPACES TO PRINTLINE.
42.4ØØØ▷          WRITE PRINTLINE.
43.ØØØØ▷          MOVE CITY TO CITY-OUT.
44.ØØØØ▷          MOVE STATE TO STATE-OUT.
45.ØØØØ▷          MOVE ZIP TO ZIP-OUT.
46.ØØØØ▷          WRITE PRINTLINE FROM CITY-STATE-ZIP-LINE.
46.2ØØØ▷          MOVE SPACES TO PRINTLINE.
46.4ØØØ▷          WRITE PRINTLINE.
47.4ØØØ▷          GO TO READ-CARD.
48.ØØØØ▷END-OF-JOB.
EDØØ4 PRESS TRANSMIT TO CONTINUE                          ⊠
```

EDT displays the lines and you see that it added the new lines on 39.5, 40.2, 40.4, 42.2, 42.4, 46.2, and 46.4. You press the XMIT key to continue.



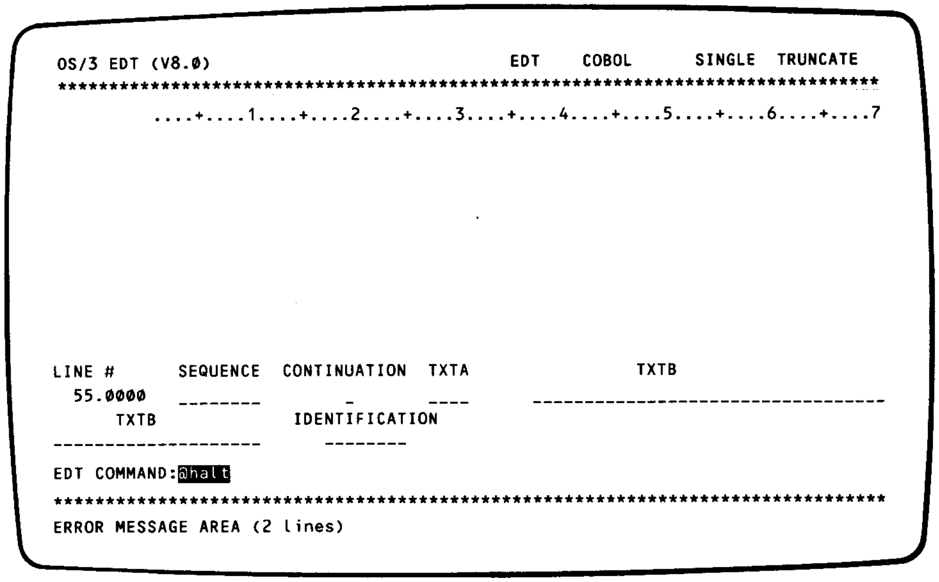
```
OS/3 EDT (V8.Ø)                                 EDT    COBOL      SINGLE  TRUNCATE
********************************************************************************
         ....+....1....+....2....+....3....+....4....+....5....+....6....+....7









LINE #      SEQUENCE  CONTINUATION  TXTA                 TXTB
  55.ØØØØ   --------       -        ----     ----------------------------------
    TXTB                IDENTIFICATION
--------------------       --------
EDT COMMAND:@w mo=labels,fil=magazine,vsn=user01
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

EDT displays another COBOL screen. You press the TAB FORWARD key to the EDT COMMAND line and enter an @WRITE command to write your program back to the disk volume. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key.



```
IS1ØØ LABELS              EXISTS; OK TO WRITE TO IT?(Y,N)Y
```

EDT reminds you that your program module already exists and double-checks if you want to overwrite it. You want to overwrite it so you respond Y for yes and press the XMIT key.



```
OS/3 EDT (V8.Ø)                               EDT    COBOL      SINGLE  TRUNCATE
********************************************************************************
          ....+....1....+....2....+....3....+....4....+....5....+....6....+....7








LINE #      SEQUENCE  CONTINUATION  TXTA                TXTB
  55.ØØØØ   ________        _       ____     __________________________________
     TXTB                IDENTIFICATION
____________________      ________
EDT COMMAND:@halt
********************************************************************************
ERROR MESSAGE AREA (2 lines)
```

On the next COBOL screen, you press the TAB FORWARD key to the EDT COMMAND line and enter an @HALT directive to end your EDT session. Then you press the TAB FORWARD key to the bottom rightmost position on the screen and press the XMIT key. EDT terminates.

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

___

(Document Title)

___ ___ ___

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

___

(Name of User)

___

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

ong line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT