

HP 3000 Computer System



Data Entry Library

Reference Manual



5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA, 95050

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the most recent date on which the technical material on any given page was altered. If a page is simply re-arranged due to a technical change on a previous page, it is not listed as a changed page: Within the manual, changes are marked with a vertical bar in the margin.

Pages	Effective Date	Pages	Effective Date
Title	Mar 1977	10-1 to 10-3	Mar 1977
ii to ix.	Mar 1977	A-1 to A-3	Mar 1977
1-1 to 1-5.	Mar 1977	B-1 to B-4	Mar 1977
2-1 to 2-5.	Mar 1977	C-1 to C-17	Mar 1977
3-1 to 3-21.	Mar 1977	D-1.	Mar 1977
4-1 to 4-5.	Mar 1977	E-1 to E-2	Mar 1977
5-1 to 5-6.	Mar 1977	F-1 to F-2	Mar 1977
6-1 to 6-5.	Mar 1977	G-1 to G-3	Mar 1977
7-1 to 7-6.	Mar 1977	H-1 to H-2	Mar 1977
8-1 to 8-3.	Mar 1977	I-1 to I-2	Mar 1977
9-1 to 9-20.	Mar 1977		

PRINTING HISTORY

New editions incorporate all update material since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover changes only when a new edition is published. If minor corrections and updates are incorporated, the manual is reprinted but neither the date on the title page and back cover nor the edition change.

First Edition. Jun 1976

Second Edition. Mar 1977

PREFACE

This manual explains how to use the Data Entry Library (DEL/3000) for the HP 3000 Series I and Series II Computer systems. Specifically, it shows you how to:

1. create and store on the system, various forms such as purchase orders, billing forms, or accounts payable/receivable records
2. display, modify or delete the forms
3. write programs that call DEL procedures to display the forms, accept and edit data transferred to the forms, and optionally store the data in data files on the system.

Your user-written programs can be written in COBOL, FORTRAN, BASIC, or Systems Programming Language (SPL).

To use this manual, you should understand the fundamental techniques of programming and be familiar with at least one of the languages described in the following manuals:

Manual Title	Part No.
BASIC/3000 Interpreter Reference Manual	30000-90026
BASIC/3000 Compiler Reference Manual	32103-90001
COBOL/3000 Reference Manual	32213-90001
FORTRAN/3000 Reference Manual	30000-90040
Systems Programming Language Manual	30000-90024

You should also be familiar with the HP 3000 Computer System, and the HP 2640A or HP 2644A terminals as described in the following documents:

Manual Title	Part No.
HP 3000 Series I MPE Commands Reference Manual	30000-90088
HP 3000 Series II MPE Commands Reference Manual	30000-90009
HP 3000 Series I MPE Intrinsics Reference Manual	30000-90087
HP 3000 Series II MPE Intrinsics Reference Manual	30000-90010
EDIT/3000 Reference Manual	03000-90012
COBOL/3000 Reference Manual	32213-90001
BASIC Interpreter Reference Manual	30000-90026
Using the HP 3000 — A Guide for the Terminal User	03000-90121
2640A Interactive Display Terminal Owner's Manual	02640-90011
2644A Mini DataStation Owner's Manual	02644-90001
2644A Mini DataStation Reference Card	5952-9950

CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
[]	An element inside brackets is <i>optional</i> . Several elements stacked inside a pair of brackets means the user may select any one or none of these elements. Example: $\begin{bmatrix} A \\ B \end{bmatrix}$ user may select A or B or neither
{ }	When several elements are stacked within braces the user must select one of these elements. Example: $\begin{Bmatrix} A \\ B \\ C \end{Bmatrix}$ user must select A or B or C.
italics	Lowercase italics denote a parameter which must be replaced by a user-supplied variable. Example: CALL <i>name</i> <i>name</i> one to 15 alphanumeric characters.
underlining	Dialogue: Where it is necessary to distinguish user input from computer output, the input is underlined. Example: NEW NAME? <u>ALPHA1</u>
superscript C	Control characters are indicated by a superscript C Example: Y ^C
<i>return</i>	<i>return</i> in italics indicates a carriage return
<i>linefeed</i>	<i>linefeed</i> in italics indicates a linefeed
...	A horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.

CONTENTS

Section I

INTRODUCING DATA ENTRY LIBRARY	1-1
Applications	1-2
Features	1-3
Using DEL	1-3
Step 1: Designing the Form	1-5
Step 2: Creating the Form	1-5
Step 3: Designing and Writing the Data Entry Entry Program	1-5
Step 4: Testing the Applications	1-5
Step 5: Running the Program for Data Entry	1-5
System Requirements	1-5

Section II

USING THE TERMINAL	2-1
Terminal Requirements	2-1
Terminal Keyboard Functions	2-1
Operating the Terminal	2-5

Section III

DESIGNING AND ENTERING FORMS	3-1
Designing the Form	3-1
Creating, Modifying, and Deleting Forms	3-6
Changing Form File Characteristics	3-9
Rules for Communicating with FORMMAINT	3-9
Creating a New Form	3-10
Laying Out the Form	3-10
Specifying Field Editing	3-13
Listing Forms in a Form File	3-15
Changing a Form	3-16
Displaying a Form	3-17
Deleting a Form	3-18
Deleting a Form File	3-18
Terminating FORMMAINT	3-19

Section IV

USING DEL PROCEDURES	4-1
How Procedures are Used	4-2
Common Procedure Characteristics	4-3

Section V

ACCESSING FORMS FROM PROGRAMS	5-1
Opening a Form File	5-1
Locating a Form in a Form File	5-3
Moving a Form to a Program Buffer	5-4
Retrieving Next Edit Specifications	5-5
Closing a Form File	5-6

Section VI

ACCESSING TERMINALS FROM PROGRAMS	6-1
Opening a Terminal File	6-1
Writing Output to a Terminal	6-2
Reading Input from a Terminal	6-2
Requesting Terminal Status	6-3
Closing a Terminal File	6-4

Section VII

CALLING EDIT PROCEDURES.	7-1
Alphabetic Data Field (ALPHAEDIT)	7-1
Alphabetic Space-Filled Field (ALPHAFILL)	7-2
Alphanumeric Data Field (ANEDIT)	7-2
Numeric Data Field (NUMRCEDIT)	7-3
Numeric Zero-Filled Field (ZEROFILL)	7-4
Numeric Range Edit (NRANGE)	7-4
Modulo 11 Check Digit Create (M11CREATE)	7-5
Modulo 11 Check Digit Verify (M11VERIFY)	7-5
Summary of DEL Edit Procedures	7-6

Section VIII

USING HIGH-LEVEL INTERFACE PROCEDURES.	8-1
Displaying Form and Reading Input	8-1
Editing Next Input Field	8-2

Section IX

WRITING PROGRAMS	9-1
COBOL Program	9-1
BASIC Program	9-12
Preparing Programs	9-12
Executing Programs	9-12
Input/Output Files	9-18
Writing User Procedures	9-18
Look-Up Tables	9-18
Detecting/Correcting Errors	9-19

Section X

CORRECTING ERRORS	10-1
------------------------------------	-------------

Appendix A

DATA AREA FORMATS	A-1
DEL/3000 Communications Area	A-1
Editing Specifications Tables	A-1

Appendix B

FORM FILE FORMAT.	B-1
----------------------------------	------------

CONTENTS (continued)

Appendix C

MPE COMMANDS SPECIFICATIONS	C-1
:HELLO	C-3
:FILE	C-7
:BYE.	C-17

Appendix D

ASCII CHARACTER SET	D-1
-------------------------------	-----

Appendix E

DATA ENTRY LIBRARY FORM LAYOUT SHEET	E-1
---	-----

Appendix F

DATA ENTRY LIBRARY EDITING SPECIFI- CATION SHEET	F-1
---	-----

Appendix G

FCHECK ERROR CODES	G-1
------------------------------	-----

Appendix H

DEL STATUS WORD SETTINGS	H-1
------------------------------------	-----

ILLUSTRATIONS

Title	Page	Title	Page
Application Program Data Flow	1-1	Form Listing Selection Display	3-17
Using DEL in an Application	1-4	Form Listing Output	3-19
Sales Transaction Form	3-2	Form Deletion Display	3-21
Sales Transaction Form Layout Sheet	3-3	Form File Deletion Verification Display	3-21
Transaction Form Editing Specification Sheet	3-5	Procedure Calls in Application Programs	4-4
Function Selection Display	3-7	Application Program Procedures in Action	5-1
Form Creation Display	3-10	COBOL Program Flowchart	9-2
Sales Transaction Report Form as Entered	3-12	COBOL Program Listing	9-3
Editing Specification Display	3-13	COBOL Program Symbol Table Map	9-10
Form File Directory Display	3-15	BASIC Program Flowchart	9-13
Form Modification Display	3-16	BASIC Program Listing	9-14



TABLES

Title	Page	Title	Page
Keyboard Functions	2-2	Edit Specifications Table Format	A-3
Form File Characteristics	3-7	Forms File Format	B-2
Function Selection Display Errors	3-8	File ID Record Format	B-3
Form Creation Display Errors	3-11	Directory Record Format	B-3
Editing Specification Display Errors	3-14	Form Definition Record Format	B-3
Summary of Procedure Parameters	4-5	Input/Edit Description Record Format	B-4
Data Characteristics	7-6	Reference Notes for Command Definitions	C-2
DEL Edit Procedure Summary	7-6	ASCII Code Chart	D-1
Error Messages Generated by FORMAINTE	10-2	DEL Status Word Settings	H-2
DEL/3000 Communications Area Format	A-2		

INTRODUCING DATA ENTRY LIBRARY

SECTION

I

The Data Entry Library (DEL/3000) helps you use an HP 2640A Interactive Display Terminal or an HP 2644A Mini DataStation to create and maintain files of formatted data on a HP 3000 Series I or Series II Computer System. First, DEL assists you in creating forms that are stored in *form files* residing on the system. Then DEL aids you in writing programs that access both the pre-defined forms and terminals. These programs can be written in COBOL, FORTRAN, BASIC, and Systems Programming Language (SPL). Typically, your program displays a form selected from a form file onto the terminal screen and allows users to enter data on the form image. You can use DEL procedures to edit the data, which you may optionally store on a data file in a format defined by your program. The form image serves as a visible guide or template for entering data. The form image is moved onto the screen through a program buffer (work area), and the data entered is transferred to the data file via this or another buffer as shown in figure 1-1.

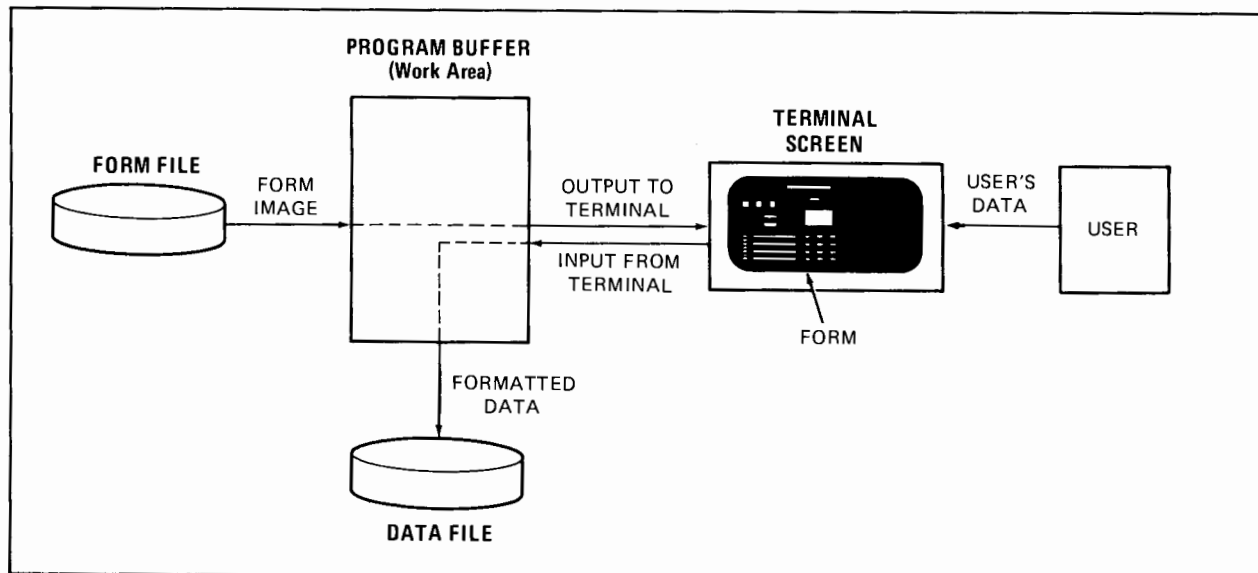


Figure 1-1. Application Program Data Flow

DEL consists of:

1. An *interactive form-maintenance program (FORMAINT)* that enables you to:
 - create the forms and store them in the form files,
 - display, modify, or delete the forms,
 - list all forms in a form file, and
 - delete entire form files.

Each form file contains one or more forms that are usually related in some way. Forms of 24 lines or less, when brought into terminal memory, are stored in one memory page and can be displayed on the terminal screen in their entirety. Larger forms occupy additional memory pages that are automatically displayed in sequence as the form data is entered or transmitted. In addition, sets of related forms can be chained together to permit their display in a given order.

2. *Form-access procedures* that allow your program to:

- open a form file,
- locate any form in that file,
- make the form available to the program by moving it into a buffer, and
- close the form file.

These procedures allow your program read-only access to the form.

3. *Terminal-access procedures* that provide read/write access to a terminal as an HP 3000 file. They permit your program to:

- open a terminal as a file,
- write output from the program buffer to the terminal,
- read input from the terminal to this or another buffer,
- request status information about the terminal, and
- close the terminal file.

These functions allow you to call up the form on your terminal screen programmatically.

NOTE

Whenever the word *terminal* appears in this manual, it applies to both the HP 2640A Interactive Display Terminal and the HP 2644A Mini DataStation, but to no other terminals, unless otherwise noted.

4. *Editing procedures* that validate the general content of user input to alphabetic, alphanumeric, and numeric fields on the forms. Your programs call these procedures, which check the appropriate input after it is entered, and then return to your program indications of whether the input data passed this edit check. For instance, if your program calls a procedure to verify that input to a field is alphabetic, but someone running that program instead enters numeric data, the editing procedure returns an error indication to your program. In addition to these procedures supplied with DEL, you may also provide your own input editing procedures to interface between your program, the forms it displays, and the user at the terminal.
5. *High-level interface procedures* that combine some of the form-access and terminal-access operations described above.

APPLICATIONS

DEL is useful for any application that displays a form or formatted report on a terminal screen, allows a user to enter or change data on the form, and edits this data as it is entered. In the fields of business and commerce, for example, applications include:

- Preparation of purchase orders, inventory transactions, invoices, and billing forms.
- Input to accounts payable and receivable systems, employee transactions, journals, and general ledgers.

- Output from management-inquiry programs.
- Low-volume generation of financial, managerial, and legal statements.
- Centralized data-processing activities in banks.

At all levels of the education field, DEL is useful for applications involving:

- On-line registration of students.
- Maintenance of student and alumni records.
- Generation of grade, progress, and status reports.
- Various payroll and accounting activities.

For scientific applications, DEL is valuable in data collection, entry, and retrieval operations. For example, it could be applied to help technicians enter data on-line into application programs that run one or more terminals.

In general, DEL is valuable for any applications that require computer interaction with forms.

FEATURES

DEL provides a simple, consistent interface with HP 2640A/2644A terminals. It is easy to use, displaying convenient interactive prompts for form creation, modification, and selection of editing routines, and eliminating the burden of transmitting escape codes and other terminal-oriented requirements when interacting with programs. (The prompts allow you to select various operations much as you would pick items from a shopping list or menu.) DEL does much of the form creation and maintenance work for you, letting you avoid the details involved in writing special programs or routines to accomplish this effort; this is particularly valuable in cases where the forms themselves are changed frequently.

DEL offers many conveniences. First, it incorporates dual-level operations — form creation/maintenance and data entry/modification — in one software package. This is useful in applications where the individual who actually uses the forms also designs them, while another person (the programmer) is concerned only with the data to appear on the forms. Two additional conveniences are the *Form Layout Sheet* (a pre-printed matrix on which you may make a preliminary sketch of your form) and the ability to produce permanent copies of forms on a line-printer. Finally, the basic field checking and editing procedures allow programs to verify the format of data as it is entered, and to detect and report format errors to the user; this, in turn, may allow him to make any necessary corrections on-line.

DEL is highly versatile. It provides the ability to call procedures from four different programming languages — COBOL, FORTRAN, BASIC, and SPL. It also allows you to draw upon all the varied functions of the HP 2640A/2644A terminals — protected fields, video high-lighting, insert/delete functions, and so forth.

USING DEL

In using DEL, you typically follow the steps outlined below and illustrated in figure 1-2.

USING THE TERMINAL

SECTION

II

If you have not yet operated the HP 2640A or HP 2644A terminal, you may read this section of the manual for an overview of major points and commonly-used functions. To learn the full capabilities and operating procedures for your terminals, however, be sure to read the appropriate owner's manual:

- *HP 2640A Interactive Display Terminal Owner's Manual*
- *HP 2644A Mini DataStation Owner's Manual*



NOTE

If you have been using the terminal regularly and fully understand how to operate it, you may skip this section and move directly to Section III.

TERMINAL REQUIREMENTS

To interface with DEL/3000, your terminal must be:

- Furnished with at least 4096 bytes (characters) of user memory.
- Equipped for line/FORMAT mode. When operating in this mode, the terminal transmits information in unprotected fields only, ignores all control characters (except record-separators (RS) characters) embedded in data entered, and generates a carriage *return* as a data-block terminator. (An *unprotected field* is a field into which you can enter data; a *protected field* is a field that you cannot alter, such as a title or column heading on a form display.)
- Operated in *block* mode, where the information you enter is transmitted to the computer in lines or blocks of characters, allowing you to edit this data before transmitting it. (When not in block mode, the terminal operates in *character* mode, where each character is transmitted as you type it.)
- Connected to the computer over an asynchronous terminal controller.

TERMINAL KEYBOARD FUNCTIONS

The HP 2640A and 2644A terminal keyboards provide keys for a wide variety of operations. Those you are most likely to use with DEL are summarized in table 2-1, where they are organized as functional groups. Others are discussed as they are used later in this manual. All are described in detail in the owner's manual.

NOTE

All discussions of keyboard functions in this manual assume that you are using the standard HP 2640A/2644A character set with upper and lower case letters.

KEY GROUP	KEY	FUNCTION
	END UNPROTECTED FIELD (on HP 2640) f3 ^C (on HP 2644)	Ends an unprotected field. Characters from present cursor position to end of this line or start of <i>next</i> unprotected field, are protected in FORMAT mode and cannot be over-written.
	FORMAT MODE (on HP 2640) f4 ^C (on HP 2644)	Places terminal in FORMAT mode (ON) so that only <i>unprotected fields</i> can be altered by either terminal user or computer. (The unprotected fields are <i>specifically defined</i> by the START UNPROTECTED FIELD and END UNPROTECTED FIELD keys. All other fields are protected in FORMAT mode.) NOTE To turn FORMAT mode off on the HP 2640, unlatch the FORMAT mode key. To do the same on the HP 2644, enter ESC X.
	BACK SPACE	Moves cursor one character to left. If cursor is in first column, it remains there.
	INSERT LINE	Rolls down line containing cursor and subsequent lines below this line, inserts blank line before cursor line, and moves cursor to first column of new blank line. Disabled in FORMAT mode.
	DELETE LINE	Deletes line containing cursor, rolls upward the following lines, and places cursor at beginning of first rolled-up line. Disabled in FORMAT mode.
	INSERT CHAR	When latched down, allows you to specify characters to be inserted at present cursor position. In FORMAT mode, operates on a field-by-field basis.
	DELETE CHAR	Deletes character at cursor position and moves all succeeding characters leftward by one column. In FORMAT mode, operates on a field-by-field basis.
	f8 ^C (on HP 2640) f8 (on HP 2644)	Special function key that returns you to Function Selection Display in FORMAIN. When entered on Editing Specification Display, generates blank edit descriptions for remaining unprotected fields on form. (See Section III.)
Additional Functions	^ ^C (Control ^) (Record separator)	Acts as end-of-data indicator.
	ESC k (Clear Line from Cursor)	Clears line from cursor position to end of current line or current unprotected field.

USING THE TERMINAL

SECTION

II

If you have not yet operated the HP 2640A or HP 2644A terminal, you may read this section of the manual for an overview of major points and commonly-used functions. To learn the full capabilities and operating procedures for your terminals, however, be sure to read the appropriate owner's manual:

- *HP 2640A Interactive Display Terminal Owner's Manual*
- *HP 2644A Mini DataStation Owner's Manual*



NOTE

If you have been using the terminal regularly and fully understand how to operate it, you may skip this section and move directly to Section III.

TERMINAL REQUIREMENTS

To interface with DEL/3000, your terminal must be:

- Furnished with at least 4096 bytes (characters) of user memory.
- Equipped for line/FORMAT mode. When operating in this mode, the terminal transmits information in unprotected fields only, ignores all control characters (except record-separators (RS) characters) embedded in data entered, and generates a carriage *return* as a data-block terminator. (An *unprotected field* is a field into which you can enter data; a *protected field* is a field that you cannot alter, such as a title or column heading on a form display.)
- Operated in *block* mode, where the information you enter is transmitted to the computer in lines or blocks of characters, allowing you to edit this data before transmitting it. (When not in block mode, the terminal operates in *character* mode, where each character is transmitted as you type it.)
- Connected to the computer over an asynchronous terminal controller.

TERMINAL KEYBOARD FUNCTIONS

The HP 2640A and 2644A terminal keyboards provide keys for a wide variety of operations. Those you are most likely to use with DEL are summarized in table 2-1, where they are organized as functional groups. Others are discussed as they are used later in this manual. All are described in detail in the owner's manual.

NOTE

All discussions of keyboard functions in this manual assume that you are using the standard HP 2640A/2644A character set with upper and lower case letters.

Table 2-1. Keyboard Functions

KEY GROUP	KEY	FUNCTION
Character Set	Alphabetic (A-Z), numeric (0-9), and special symbols (=, #, \$, and so forth).	Similar to standard typewriter key operations; enters character selected, in upper or lower case.
	ESC	Generates ASCII escape character; transfers from normal operational mode to allow you to enter commands for programmable terminal functions. (For instance, <i>ESC U</i> produces a NEXT PAGE command.) See Appendix D for character set summary and owner's manual for further details.
	CNTL	When held down while any alphabetic key or @, [, \,], ^, -, /, {, , } , ~, or DEL key is pressed, converts the character code for that key to an ASCII control code. (See Appendix D for character set summary.)
	LINE FEED (on HP 2640) or J ^c (on HP 2644)	Moves cursor down one line. If cursor is on last line displayed, moves entire display up one line and places cursor in new last line.
	RETURN	Returns cursor to beginning of current line. When entered at end of line, also generates a line-feed.
Numeric and Display Control Note: See owner's manual for many other keys in this group.	↑	Moves cursor up one line. If cursor is in top line of display, wraps cursor around to bottom line.
	↓	Moves cursor down one line. If cursor is in bottom line, wraps cursor around to top line.
	→	Moves cursor one column to right. If cursor is in last column, wraps cursor around to first column of next line. If cursor is in last position of display, wraps cursor around to first position.
	←	Moves cursor one column to left. If cursor is in first column, wraps it around to last column of above line. If cursor is in first position of display, wraps it around to last position.
	↖	Moves cursor to first unprotected (Home) position of display.
Edit and Control Group	RESET TERMINAL	Sets terminal to its initial state when power was turned on: clears display and memory, moves cursor to Home position, shuts off programmable functions. NOTE When using this function in a program, use extreme caution and be sure you accurately anticipate the results.
	BREAK	Requests a system break, returning control to MPE.
	DISPLAY FUNCTIONS	Disables all escape codes and control functions (except <i>return</i>) entered or received. With 128-character Roman character set, displays escape codes and control functions

Table 2-1. Keyboard Functions (Continued)

KEY GROUP	KEY	FUNCTION																																																																																				
		associated with an entry as an aid in debugging. (See owner's manual.)																																																																																				
	BLOCK MODE	When latched down, places terminal in <i>block mode</i> .																																																																																				
	REMOTE	When latched down, places terminal in remote (on-line) operating mode for communicating with the computer.																																																																																				
	TAB	(In FORMAT mode), moves cursor to start of next unprotected field; disregards normal horizontal tab stops.																																																																																				
	ENTER	In DEL's FORMAIN Program, sends entire form to form file. In response to DEL application program prompts, transfers all input data in unprotected field to program's input buffer.																																																																																				
	ENHANCE DISPLAY (on HP 2640) f1^c (on HP 2644) NOTE The superscript c indicates the CNTRL function — thus, you must hold down the CNTL key while pressing f1.	When followed by one of the letters A through O, indicates display of one of 15 possible combinations of half-bright, underline, inverse video (black letters on white background), and blinking characters. When followed by commercial at-sign (@) entry, cancels previously-selected combination. The combinations possible are listed in this chart: <table><tr><th></th><th>@</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th><th>G</th><th>H</th><th>I</th><th>J</th><th>K</th><th>L</th><th>M</th><th>N</th><th>O</th></tr><tr><td>Half Bright</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>Underline</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>Inverse Video</td><td></td><td></td><td></td><td>X</td><td>X</td><td></td><td>X</td><td>X</td><td></td><td></td><td>X</td><td>X</td><td></td><td></td><td>X</td><td>X</td></tr><tr><td>Blinking</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td></tr></table> In the above chart, X indicates that the feature is displayed. For example, to start a field of blinking, underlined characters on the HP 2640, press the ENHANCE DISPLAY key followed by the E key. To terminate the blinking underlined field, press the ENHANCE DISPLAY key followed by the @ key. You could use these keys to visually distinguish between protected versus unprotected fields, or to bring certain fields to the attention of the person entering data on the form. NOTE On the HP 2644 terminal you must hold down CNTL while pressing the f1 key. Thus, to start a blinking underlined field on that terminal, hold CNTL down, press f1, release CNTL and strike E.		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	Half Bright									X	X	X	X	X	X	X	X	Underline					X	X	X	X					X	X	X	X	Inverse Video				X	X		X	X			X	X			X	X	Blinking		X		X		X		X		X		X		X	
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O																																																																						
Half Bright									X	X	X	X	X	X	X	X																																																																						
Underline					X	X	X	X					X	X	X	X																																																																						
Inverse Video				X	X		X	X			X	X			X	X																																																																						
Blinking		X		X		X		X		X		X		X		X																																																																						
START UNPROTECTED FIELD (on HP 2640) f2^c (on HP 2644)	Starts an unprotected field. In FORMAT mode, characters from present cursor position to end of this field or end of this line are unprotected and can be overwritten. (These characters are normally spaces.)																																																																																					

Table 2-1. Keyboard Functions (Continued)

KEY GROUP	KEY	FUNCTION
	END UNPROTECTED FIELD (on HP 2640) f3 ^C (on HP 2644)	Ends an unprotected field. Characters from present cursor position to end of this line or start of <i>next</i> unprotected field, are protected in FORMAT mode and cannot be over-written.
	FORMAT MODE (on HP 2640) f4 ^C (on HP 2644)	Places terminal in FORMAT mode (ON) so that only <i>unprotected fields</i> can be altered by either terminal user or computer. (The unprotected fields are <i>specifically defined</i> by the START UNPROTECTED FIELD and END UNPROTECTED FIELD keys. All other fields are protected in FORMAT mode.) NOTE To turn FORMAT mode off on the HP 2640, unlatch the FORMAT mode key. To do the same on the HP 2644, enter ESC X.
	BACK SPACE	Moves cursor one character to left. If cursor is in first column, it remains there.
	INSERT LINE	Rolls down line containing cursor and subsequent lines below this line, inserts blank line before cursor line, and moves cursor to first column of new blank line. Disabled in FORMAT mode.
	DELETE LINE	Deletes line containing cursor, rolls upward the following lines, and places cursor at beginning of first rolled-up line. Disabled in FORMAT mode.
	INSERT CHAR	When latched down, allows you to specify characters to be inserted at present cursor position. In FORMAT mode, operates on a field-by-field basis.
	DELETE CHAR	Deletes character at cursor position and moves all succeeding characters leftward by one column. In FORMAT mode, operates on a field-by-field basis.
	f8 ^C (on HP 2640) f8 (on HP 2644)	Special function key that returns you to Function Selection Display in FORMAIN. When entered on Editing Specification Display, generates blank edit descriptions for remaining unprotected fields on form. (See Section III.)
Additional Functions	^ ^C (Control ^) (Record separator)	Acts as end-of-data indicator.
	ESC k (Clear Line from Cursor)	Clears line from cursor position to end of current line or current unprotected field.

OPERATING THE TERMINAL

To prepare your terminal for use, follow these steps:

1. Assure that the REMOTE latching key is not depressed; thus, the terminal is set for *off-line (local)* operation.
2. Set the A.C. POWER Switch, located on the terminal's rear panel, to the ON position. After a 15-second warm-up period, the terminal enters its initial operating state with the display screen and memory cleared, the cursor in the upper-left corner (*Home* position) of the screen, and all programmable functions turned off.
3. At your option, press the TEST key to validate the operation of the terminal's memory, firmware, and display. Generally, if the terminal emits an audible "beep" and a test pattern appears that is similar to those described in the Self-Test section of your owner's manual, the terminal is working properly. This pattern also denotes the character sets available on your terminal. (See owner's manual.) Press the RESET TERMINAL key to clear the display.

NOTE

If the TEST function does not work properly and the cursor still does not appear, set the A.C. POWER switch to OFF and call an HP service representative. Do not try to operate the terminal again until it has been repaired.

4. Set the following keys and switches as indicated below:

Key/Switch	Position
BLOCK MODE	OFF (unlatched)
AUTO L.F. (Line Feed)	OFF (unlatched)
DUPLEX	FULL
PARITY	NONE
BAUD RATE (Speed)	Any setting desired

To initiate communication with the computer, latch down the REMOTE key to set your terminal to the on-line (remote) mode and log-on as directed in either:

- *MPE Commands Reference Manual*, or
- *Using the HP 3000*

This procedure requires use of the MPE command :HELLO, discussed in detail in the *MPE Commands Reference Manual* and summarized in Appendix C of this manual.

To terminate communication with the computer, enter the :BYE command as directed in the above manual and summarized in Appendix C.

To shut off your terminal:

1. Press the RESET TERMINAL key to re-initialize the terminal.
2. Set the A.C. POWER switch to the OFF position, turning off the power.

DESIGNING AND ENTERING FORMS

SECTION

III

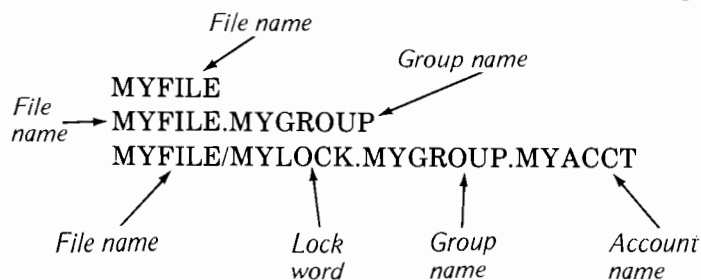
The first steps in any DEL/3000 application involve designing and entering the form (or forms) that the application will use. These steps and those that follow are best illustrated by an example. Suppose, for instance, that you are preparing an application that enables a clerk in a construction supply store to interactively enter data onto a sales transaction form using an HP 3000 computer system. The application program is to display the form on the terminal screen showing the following information:

- Form title: "SALES TRANSACTION"
 - Date of transaction
 - Transaction number
 - Customer identification number
 - Salesman identification number
- } (Six-digit numeric code)
- Billing recipient
 - Product number (four-digit numeric code)
 - Product description
 - Price per unit
 - Quantity sold

DESIGNING THE FORM

If the clerk were manually entering the data upon a paper form for this application, that form would resemble the one shown in figure 3-1. When you adapt this form for terminal display and enter it in the computer, you will need to know exactly where each field begins and ends, and what kind of editing requirements, if any, apply to each unprotected field. (In figure 3-1, the unprotected fields are circled.) You can determine the precise location of each field by laying out the form on the *DEL/3000 Form Layout Sheet* provided in Appendix E of this manual. For the sales transaction example, the layout sheet shown in figure 3-2 might be used. On this sheet, you should enter these elements:

- **Form File Name.** This is the MPE formal file designator of the file on which your form will reside. In addition to the file name, it may also include a file lockword, a group name, and an account name. The names and lockword may each contain up to eight alphanumeric characters, beginning with a letter. Within the total entry, you must also include any appropriate delimiting periods and slash-marks. The total entry must not exceed 35 characters. See *MPE Commands Reference Manual* for further information on formal file designators. Examples are:



SALES TRANSACTION

DATE: 09/06/76 TRANSACTION NO: 000047

CUSTOMER ID: 004272 BILLED TO: RAY C. SMITH

SALESMAN ID: 006307 AJAX CONSTRUCTION CO.

3671 POTTER DR.

SPRINGVALE, CA 96127

PRODUCT NO.	DESCRIPTION	UNIT PRICE	QUANTITY
7021	GALV. IRON NAILS #8	0.50	30
7025	GALV. IRON NAILS #10	0.60	30
8011	RE-ENFORCING WIRE	2.35	15

Figure 3-1. Sales Transaction Form

On the Form Layout Sheet for the sales transaction report, the form file name SALESFIL.PUB is used. This specifies a file in the public group of your log-on account.

- **Form Name.** This is the name by which DEL will recognize and access your form. This name can include up to 16 alphanumeric characters, beginning with either a letter or a digit. Within the form file, this name must be unique. Examples are:

MYFORM
 ACCOUNTSREC
 ACCOUNTSPAY
 GENLEDGER1
 FORM22A
 3 FORM

On the layout sheet for the Sales Transaction Report, the form name SALESFORM is used.

3-3

[illegible][illegible]

Figure 3-2. Sales Transaction Form Layout Sheet

- *Chain To Form Name.* This entry allows you to name any other form to which the form you are now designing relates. When specified during the form design process, this entry permits your application program to automatically display the forms sequentially. Because the form for the sales transaction report is not related to any other form, this entry is left blank in the example.
- *Page Number.* If your form contains 24 lines or less, it will occupy only one page of terminal memory and will fit completely onto the terminal screen; in this case, you can indicate "Page 1 of 1" in the upper-right corner of the Form Layout Sheet as in the example. If, however, you are producing a larger form that requires additional memory pages, you may continue it onto one or more additional layout sheets, indicating the particular page number and total number of pages in this entry, such as "Page 2 of 3". When the form is entered and displayed, the terminal automatically presents each successive page on the screen in the sequence required.

To locate the individual fields for a form, simply enter these fields in the lines and columns desired on the layout sheet. To indicate the contents of unprotected fields, you may use any convention you desire. In the sales transaction report example, Xs are used to show alphanumeric fields and Zs are used for numeric fields.

Immediately after you enter your form via FORMAINTE, you must specify any editing requirements that apply to the unprotected fields on your form. At that time, it will be helpful if you have all the editing information for each field already at hand. You can define this information by filling out the *Editing Specification Sheet* provided in Appendix F, listing all requirements that apply to each edited field. For the sales transaction example, a sample specification sheet appears in figure 3-3. This sheet allows you to specify the following items:

- *Field Name.* Enter the name or description of the unprotected field. The first field named in the example is the "Month" portion of the DATE entry.
- *Length.* Enter the length of the unprotected field. In the example, the "Month" entry is two digits long.
- *Edit Type.* Enter the type of editing you desire for this field. You may wish to specify a combination of different types. For the "Month" entry in the example, the application requires that numeric data only appears in the entry, that the data is right-justified and the field is filled with zeros to the left of the data, and that the data falls within a given numeric range (01 through 12).
- *Edit Procedure Name.* Enter the name of the procedure that will edit this field when your application program runs. You may name more than one procedure, selecting from those furnished by DEL, procedures written by yourself, or both. (DEL procedures and the functions they perform are described in detail in Section VII.) Each procedure name may include up to 15 of any characters legal in the language used for your application program. In the specification sheet for the example, the CNRANGE procedure is used for the "Month" entry to provide numeric/zero-fill editing and range-checking.
- *Test Flag # Before Edit.* DEL accesses 16 one-bit flags that are all cleared to zero each time a new form is accessed. Both DEL and user-supplied edit procedures may set these flags, which may be tested by your application program to allow conditional editing or logical editing between two or more fields. These flags are numbered 1 through 16. If the procedure named in the previous column is to be used conditionally or is to be modified by a previously-invoked edit procedure, specify the related flag to be tested. (No flags are used in the sales transaction example.)
- *After Edit, Set Flag #.* If the procedure named is to set one of the flags when it edits the data, specify which flag this is.

DATA ENTRY LIBRARY EDITING SPECIFICATION SHEET

Field Name	Length	Edit Type	Edit Procedure Name	Test Flag# Before Edit	After Edit Set Flag#	"Same as" Flag#	"Opposite" Flag#	Range Check		Look-up Procedure File Name	Characters
DATE (MONTH)	2	NUMERIC	CNRANGE					Low Value	High Value		
		ZERO-FILL									
		RANGE-						01	12		
		CHECK									
DATE (DAY)	2	NUMERIC	CNRANGE								
		ZERO-FILL									
		RANGE-						01	31		
		CHECK									
DATE (YEAR)	2	NUMERIC	CNRANGE								
		ZERO-FILL									
		RANGE-						70	79		
		CHECK									
TRANSACTION	6	NUMERIC	CNUMRCENT								
CUSTOMER I.D.	6	NUMERIC	CNUMRCENT								
SALESMAN I.D.	6	NUMERIC	CNUMRCENT								
PRODUCT NO.	4	NUMERIC	CNUMRCENT								
UNIT PRICE (DOLLARS)	4	NUMERIC	CZEROFILL								
		ZERO-FILL									
UNIT PRICE (CENTS)	4	NUMERIC	CZEROFILL								
		ZERO-FILL									
QUANTITY	4	NUMERIC	CNUMRCENT								

Figure 3-3. Transaction Form Editing Specification Sheet

- *Same As Flag #*. If the result of this edit must be compared with that of a prior edit and a flag is to be set if the results match, specify which flag this is.
- *Opposite Flag #*. If the result of this edit must be compared with that of a prior edit and a flag is to be set if the results do not match, specify which flag this is.
- *Range Check, Low Value and High Value*. If a range-check applies to this field, enter the lowest and highest value permitted in the field. Each value may be up to 16 characters long.
- *Look-Up Procedure File Name*. If the field is edited by a user-written procedure that looks up information in a file, enter the name of that file. Such a procedure might, for instance, scan a table to determine a tax percentage or some other factor. The file name entry may be required for linkage by this procedure, depending on how the procedure is coded. This entry can include file name and optional lockword, group name, and account name.
- *Characters*. If the edit procedure does not perform range-checking or a file look-up, you may enter in this column up to 32 characters of data to be used in any way that your procedure requires. These characters are written exactly as shown into the record containing the edit specifications on your form file. You might use this feature, for example, for an edit procedure that verifies that the data entered is any of eight four-digit account numbers; these numbers could be stored as this set of 32 characters, making it unnecessary for the editing procedure to perform an extensive file search for this data.

CREATING, MODIFYING, AND DELETING FORMS

Once you have designed your form and have a clear idea of the editing specifications required, you are ready to create the form in the system. When the form is created, you can display, modify, or delete it, and perform various other operations. For all of these functions, you must run the FORMMAINT program by logging-on and entering:

```
:RUN FORMMAINT.PUB.SYS
```

If you have logged-on at a terminal other than an HP 2640/2644, FORMMAINT prints the following message and immediately terminates:

```
INPUT DEVICE IS NOT A 2640
```

Otherwise, FORMMAINT prints the message:

```
DEPRESS BLOCK MODE KEY
```

Latch the BLOCK MODE key down, placing the terminal in block mode. FORMMAINT now presents the message HP 32206A.uu.ff FORM MAINTENANCE on the terminal screen, followed by the *Function Selection Display* shown in Figure 3-4. (In this message, uu and ff are digits indicating the current software update level and fix level, respectively, assigned by HP.)

Enter the name of your form file in the input field indicated by the present cursor position. When the entire screen contents are transmitted to FORMMAINT as directed below, FORMMAINT will create the form file as a permanent file on disc with the characteristics listed in table 3-1. If the file name indicates an old (existing) file, FORMMAINT will access that file. In the example in figure 3-4, the name of the new file SALESFIL.PUB, indicated on the layout sheet, is entered.

Cursor position

HP32206A. 00. 00 FORM MAINTENANCE (C) HEWLETT-PACKARD CO. 1976

Enter the name of your form file here
 and select one of the following functions by entering an X in front of the
 desired function.

- ☒ DEFINE A NEW FORM
- ☐ LIST FORM FILE DIRECTORY
- ☐ MODIFY AN EXISTING FORM
- ☐ DISPLAY AN EXISTING FORM
- ☐ DELETE AN EXISTING FORM
- ☐ DELETE THE FORM FILE
- ☐ EXIT FORMMAINT

Figure 3-4. Function Selection Display

Table 3-1. Form File Characteristics

CHARACTERISTIC	ASSIGNED VALUE
Record size	64 bytes (characters)
Record type	Fixed-length, ASCII format
Blocking factor	8 records per block
File size	10,000 logical records
Number of extents allowed	10
Number of extents initially allocated	1
Access type	Exclusive

NOTE

Because FORMMAINT grants a user *exclusive access* to a form file, two or more users cannot use FORMMAINT to operate on the same form file simultaneously. For the same reason, no one can run an application program to enter data via a form file while someone else is using FORMMAINT to operate on that same file.

Now select the function you desire, as follows:

1. Use the TAB key to move the cursor to the input field that indicates the operation you desire.
2. Enter the character X (upper or lower case) in the input field.
3. Press the ENTER key to transmit your entries to FORMAIN'T and begin the function you selected.

In the example in figure 3-4, the function DEFINE A NEW FORM is selected.

If you specified more than one function, the function nearest the top of the display is performed and all others are ignored.

If you did not make the required entries on the Function Select Display, these errors will be reported to you when you press ENTER, via the message shown in table 3-2. Following the message output, FORMAIN'T waits for you to correct the error and press ENTER again.

NOTE

Several errors that you may encounter when using FORMAIN'T are noted throughout this manual; all errors possible are discussed in detail in Section X. In general, if the error is caused directly by your input to a particular field, that field will blink on the display.

Table 3-2. Function Selection Display Errors

MESSAGE	CAUSE
FILE NAME REQUIRED	You did not enter a form file name.
FORM FILE NAME INVALID	You entered an invalid form file name (perhaps not beginning with a letter or containing too many characters).
FILE IS NOT A FORMS FILE	You specified an existing file that is not a form file.
FILE IS INACCESSIBLE	Another user is presently using the form file.
FILE CANNOT BE OPENED ERROR CODE=nnnn	FORMAIN'T cannot open the form file you specified. In this message, <i>nnnn</i> indicates an error code return through FORMAIN'T by the FCHECK intrinsic, as discussed in <i>MPE Intrinsic Reference Manual</i> and Appendix G of this manual. See that discussion for the meaning of this code.
NO FUNCTION SELECTED	You did not select any function from the list displayed.

CHANGING FORM FILE CHARACTERISTICS

Although you cannot change the record size, record type, and access type for any form file, you can re-specify certain other characteristics for new form files. These characteristics are: blocking factor, file size, number of disc extents allowed, and number of extents initially allocated. You may, for instance, want to conserve file space by specifying a form file shorter than that normally provided. To do this, use the MPE :FILE command to specify a new file size, as discussed in *MPE Commands Reference Manual*. For instance, to assign a file size of 6,000 records with a blocking factor of 16 records per block for SALESFIL, saving this file in the MPE permanent file domain, enter:

```
:FILE SALESFIL.PUB;REC= -64,16,F,ASCII;DISC= 6000;SAVE
```

NOTE

It is usually difficult to predict the file space actually needed for a particular form file because many factors apply. Nonetheless, if you wish to alter the size of your file, consider these points:

- One tenth of the total file space is always allocated for the form file directory that contains the names and locations of the forms in the file. Thus, a 10,000-record file requires 100 directory entries and contains a maximum of 100 forms.
- One form file edit record is required for every edit procedure named for every edited field.
- Blank areas on the form that exceed seven characters are removed when the form specifications are stored.

RULES FOR COMMUNICATING WITH FORMAINIT

When responding to the prompts in the Function Selection Display as well as those for other FORMAINIT displays, remember these general rules:

1. To skip to a new prompt (input field), press the TAB key.
2. To continue with the normal progression of a function, press the ENTER key.
3. To exit from a function and re-display the Function Selection Display, use:
 - The f8 key, on the HP 2644.
 - The f8 key with CNTL depressed (f8^c) on the HP 2640.
4. To correct an entry, re-position the cursor and use the DELETE CHAR key or type over the existing entry.

In general, if you enter data that equals the length of an input field for a prompt, the cursor skips to the next input field. The next character you enter appears in this field.

CREATING A NEW FORM

To create a new form, enter an X beside the DEFINE A NEW FORM entry in the Function Selection Display and press ENTER. In response, FORMAINT presents the Form Creation Display shown in figure 3-5.

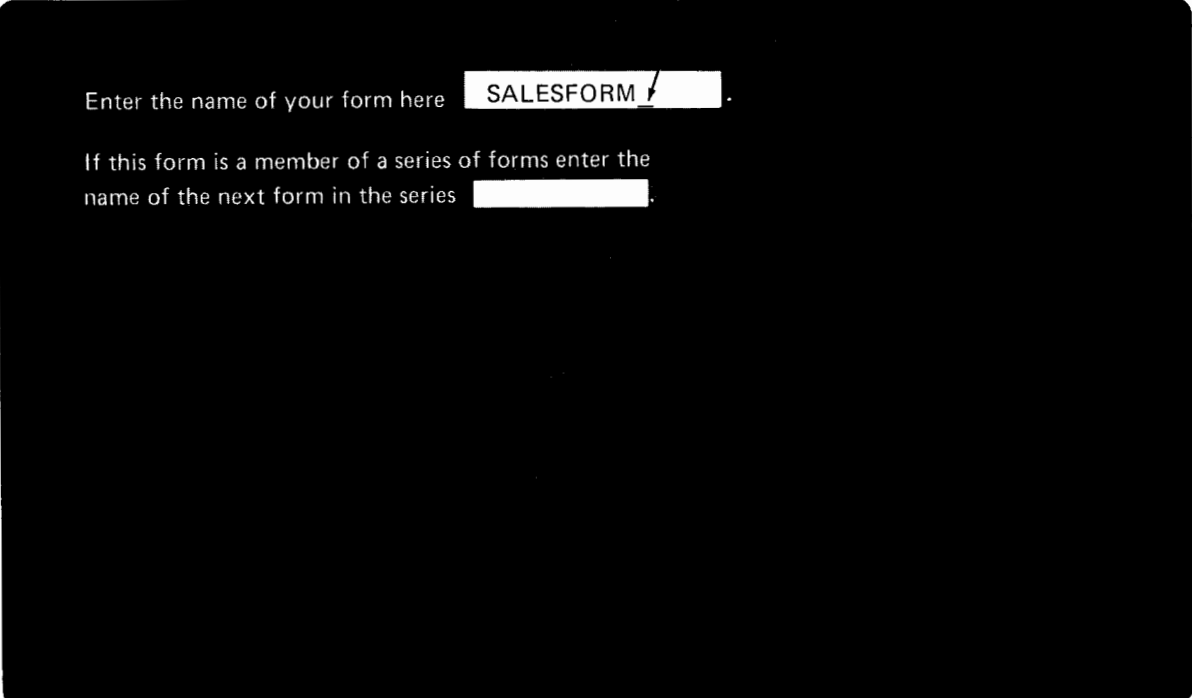
Respond to the first prompt in this display by entering the name you wish to assign your new form. You may copy this from your Form Layout Sheet. No other form of this name is permitted in this form file. The name SALESFORM is entered in response to this prompt in figure 3-5.

If you wish to relate (chain) this form to any other form, respond to the second prompt by entering the name of the other form; this will enable your application program to automatically display both forms in sequence. Otherwise, skip this prompt.

Press ENTER to transmit your responses to FORMAINT. At this point, FORMAINT clears the screen, permitting you to lay out the form on the screen. If you have made an error in any response, it is reported to you by one of the messages in table 3-3.

LAYING OUT THE FORM. To lay out the form on the terminal screen, simply enter characters where you wish them to appear, using the Form Layout Sheet as a guide. Use the cursor-positioning keys to skip rapidly about the form. Use the enhance-display capabilities to indicate areas of the form to be filled-in by the person running your application program. Use the terminal's line-drawing set, if available, to include any horizontal or vertical lines or boxes on the form.

*Present cursor position;
press TAB to skip to next
prompt (if desired).*

A screenshot of a terminal window titled 'Form Creation Display'. The background is black with white text. The first prompt is 'Enter the name of your form here' followed by a white rectangular input field containing the text 'SALESFORM'. A cursor is visible at the end of the text in the input field. The second prompt is 'If this form is a member of a series of forms enter the name of the next form in the series' followed by an empty white rectangular input field. An arrow points from the explanatory text above to the first input field.

Enter the name of your form here SALESFORM

If this form is a member of a series of forms enter the name of the next form in the series

Figure 3-5. Form Creation Display

Table 3-3. Form Creation Display Errors

MESSAGE	CAUSE
FORM FILE IS FULL	No space is available in the form file for the new form. FORM-MAINT returns to the Function Selection Display, allowing you to specify a new form file or function.
FORM ALREADY EXISTS IN <i>filename</i>	You specified a form that already exists in the form file named <i>filename</i> . FORM-MAINT waits for you to enter a new form name.
FORM NAME REQUIRED	You failed to enter a form name. FORM-MAINT waits for you to enter a name.

NOTE

If you are not certain what graphic capabilities are furnished with your terminal, press the TEST key and examine the resulting test pattern display. (See Section II.) (All terminals are equipped with the inverse-video capability.)

When entering your form, use the following guide lines:

1. To define an unprotected field:
 - a. Press the START UNPROTECTED FIELD key (on the HP 2640) or enter f2^c (on the HP 2644) to begin the field.
 - b. Enter the characters in the field; these are typically blanks (spaces).
 - c. Press the END UNPROTECTED FIELD key (on the HP 2640) or enter f3^c (on the HP 2644) to terminate the field.

NOTE

FORM-MAINT does not allow unprotected fields that extend beyond the end of a line. If you attempt to continue such a field onto a second line, FORM-MAINT converts it into two separate fields.

2. To use the display enhancement features of the terminal:
 - a. Press the ENHANCE DISPLAY key (on the HP 2640) or enter f1^c (on the HP 2644), and then press the key for the enhancement you wish (blinking field, inverse video, or so forth) to begin the enhancement. (See Section II.)
 - b. Enter the characters in the field.
 - c. Press the ENHANCE DISPLAY key (on the HP 2640) or enter f1^c (on the HP 2644), and then press the @ (commercial at-sign) key to turn-off the enhancement.

NOTE

To help keep track of where each unprotected field begins, users often start a display enhancement for an unprotected field at the same time they start that field. Similarly, they typically terminate the display enhancement and unprotected field simultaneously. For instance, on an HP 2640, a user might begin a field by pressing the ENHANCE DISPLAY key followed by the key for his desired enhancement, and then pressing the START UNPROTECTED FIELD key. He could then observe the spaces making up this field on the screen as he enters them.

3. To indicate that the form is complete, enter the $\wedge$ (CONTROL $\wedge$) key, producing a record-separator. This indicates the end of the form and, on most terminals, displays the following characters on your screen:

R_S

FORMAINT arbitrarily terminates the form after encountering a sequence of twenty consecutive blank lines. Any information appearing after such a sequence is ignored.

4. To transmit the form to your form file, press ENTER.

For the Sales Transaction Report in the continuing example, you could enter the form as shown in figure 3-6. The unprotected fields are indicated by the inverse video (white on black) entries. The unprotected area labeled *BILLED TO*: is actually composed of four one-line unprotected fields for the mailing address of the person billed. Five unprotected fields appear under each of the headings *PRODUCT NO.*, *DESCRIPTION*, *UNIT PRICE*, and *QUANTITY* to allow for recording the sale of five items.

SALES TRANSACTION REPORT

DATE: / /

CUSTOMER ID:

SALESMAN ID:

TRANSACTION NO:

BILLED TO:

PRODUCT NO.	DESCRIPTION	UNIT PRICE	QUANTITY
 	 	 	
 	 	 	
 	 	 	
 	 	 	
 	 	 	

R_S

Figure 3-6. Sales Transaction Report Form As Entered

SPECIFYING FIELD EDITING. When the form is stored in the form file, FORMAINTE scans the form for unprotected input fields. As the first unprotected field is located, the line containing it and the most recently preceding non-blank line are displayed at the top of the screen, with an arrow pointing to the field. Below these two lines, FORMAINTE presents the *Editing Specification Display* (as shown in figure 3-7).

- If you do not want to edit any unprotected fields on the form, press the f8 key on the HP 2644 or f8^C keys on the HP 2640 to terminate the form-creation function and return the Function Selection Display to the screen.
- If you do not plan to edit the current field but wish to edit others, enter X at the first prompt in the display and press ENTER to move the field-indicating arrow to the next unprotected field; continue this procedure until the field to be edited is reached.
- If you want to edit the current field, use the TAB key to skip to the applicable prompts. These prompts allow you to enter the following options, in the order they appear on your Editing Specifications Sheet: *Edit Procedure Name*, *Test Flag # Before Edit*, *After Edit Set Flag #*, *Same As Flag #*, *Opposite Flag #*, *Range Check Low and High Value*, *Look-Up Procedure File Name*, and *Characters*. To request editing, you must supply an Edit Procedure Name; you may enter or omit any of the other options according to your editing requirements. To transmit the current screen contents to FORMAINTE, press ENTER. This returns the cursor to the beginning of the Editing Specification Display, allowing you to optionally specify additional procedures for this field. When

Arrow indicating field to be edited

Second line from form.

First line from form.

Cursor

SALES TRANSACTION REPORT

DATE: / / TRANSACTION NO:

^

If no editing is required or all edits for this field have been specified enter an X here .

The edit procedure name is CZEROFILL_ ←

Test flag # before performing edit. After edit set flag # and it must be the same as flag # or opposite flag # .

For range check editing the low value is
and the high value is .

For file look-up procedures the file name is

If the edit is not a range check nor a file look-up you may enter up to 32 characters in this space for use by the edit procedure.

Figure 3-7. Editing Specification Display

you have specified all editing for this field, enter X at the first prompt and press ENTER to move to the next unprotected field. When you wish to terminate entry of editing specifications, press the f8 key (on the HP 2644) or enter f8° (on the HP 2640). When you have defined the last edit specification for the last unprotected field on the form, this termination occurs automatically. At this time, all edit specifications are copied to the form file, and the Function Selection Display returns to the screen with the current form file name appearing at the prompt that requests entry of a form file name. You may now select another function for this file. Alternatively, you may select another form file by entering its name; in this case, FORMAINTE closes the current file as the form file name and opens the new one.

NOTE

When you press the ENTER key, all entries or responses to prompts presently appearing on your screen are transmitted to FORMAINTE. Thus, if you are specifying a second editing procedure for a field and old entries for the first procedure still appear on your screen following the cursor, clear these old entries if you do not want them to apply to the second procedure. Do this by pressing the CLEAR DISPLAY key after the last entry you do wish to apply. Then, press ENTER to transmit the entries on the screen.

When you terminate the editing process, FORMAINTE detects any erroneous entries and reports them as indicated in table 3-4.

Table 3-4. Editing Specification Display Errors

MESSAGE	CAUSE
FORM DEFINITION EXCEEDS SYSTEM CAPABILITY (Series I message)	You defined a form which contains more than 216 characters on one line. This exceeds Series I system capability. (If the form is defined on a Series II System, however, it will be acceptable to a Series I.)
FORM DEFINITION TOO LARGE FOR <i>filename</i>	You defined a form that is too long for the form file. (This may occur particularly if you specified a form file shorter than the default length of 10,000 records.) FORMAINTE returns the Function Selection Display to the screen.
EDIT PROCEDURE NAME IS INVALID	You entered an edit procedure name that does not begin with a letter or that contains a special character, or you omitted a procedure name where one is required. The cursor returns to the beginning of the Editing Specifications Display, allowing you to correct the error.
FLAG # IS INVALID	You entered a number other than 1 through 16 or a blank in response to a prompt for a flag. FORMAINTE causes the invalid entry to blink and waits for you to correct the error.
SELECT ONE — RANGE CHECK, FILE LOOK-UP OR EDIT DEFINED DATA	You specified more than one of the following entries for an input field: <i>Range Check, Look-Up Procedure File Name, or Characters</i> (for edit-defined data). FORMAINTE waits for you to correct the error.

NOTE

Because of the security restrictions provided the form file by FORMAINTE, you may neither rename the form file (MPE :RENAME command) nor transfer this file to a new group or account and still retain access to this file. If you attempt to do this, the message, FILE IS NOT A FORMS FILE appears when you attempt to access the form via FORMAINTE. (OPENFORM returns a value of minus one to the DEL status word.)



LISTING FORMS IN A FORM FILE

Each form file contains a directory that indicates the names of all forms residing in that file, plus other information about these forms that must be maintained and used by DEL. The standard content and format of form file directories is described in Appendix B. You can request FORMAINTE to display the names of the forms appearing in the directory for the current form file, plus the date and time these forms were created. To do this, enter X beside the LIST FORM FILE DIRECTORY prompt in the Function Selection Display and press ENTER. The directory display appears as shown in figure 3-8; this example, requested for the file named SALESFIL, shows that this file contains two forms, SALESFORM and COSTFORM.

To clear the screen and return to the Function Selection Display, press ENTER; alternatively, you may press f8 (on the HP 2644) or f8^c (on the HP 2640).

FORM NAME			CREATION DATE			TIME		
FORM NAME			CREATION DATE			TIME		
SALESFORM			4/16/76			10:58		
COSTFORM			4/16/76			12:16		

Figure 3-8. Form File Directory Display

CHANGING A FORM

To change a form already defined in a form file, enter X at the MODIFY AN EXISTING FORM entry in the Function Selection Display and press ENTER. FORMAINTE presents the Form Modification Display as shown in figure 3-9, allowing you to specify this form. If the form is chained to another form, and you wish to chain it to a different form, enter the name of the new second form in the second prompt.

CAUTION

WHEN YOU REQUEST THE MODIFY OPERATION, ALL EDITING SPECIFICATIONS THAT APPLY TO THE CURRENT FORM ARE AUTOMATICALLY DESTROYED. BEAR THIS IN MIND BEFORE REQUESTING THIS FUNCTION.

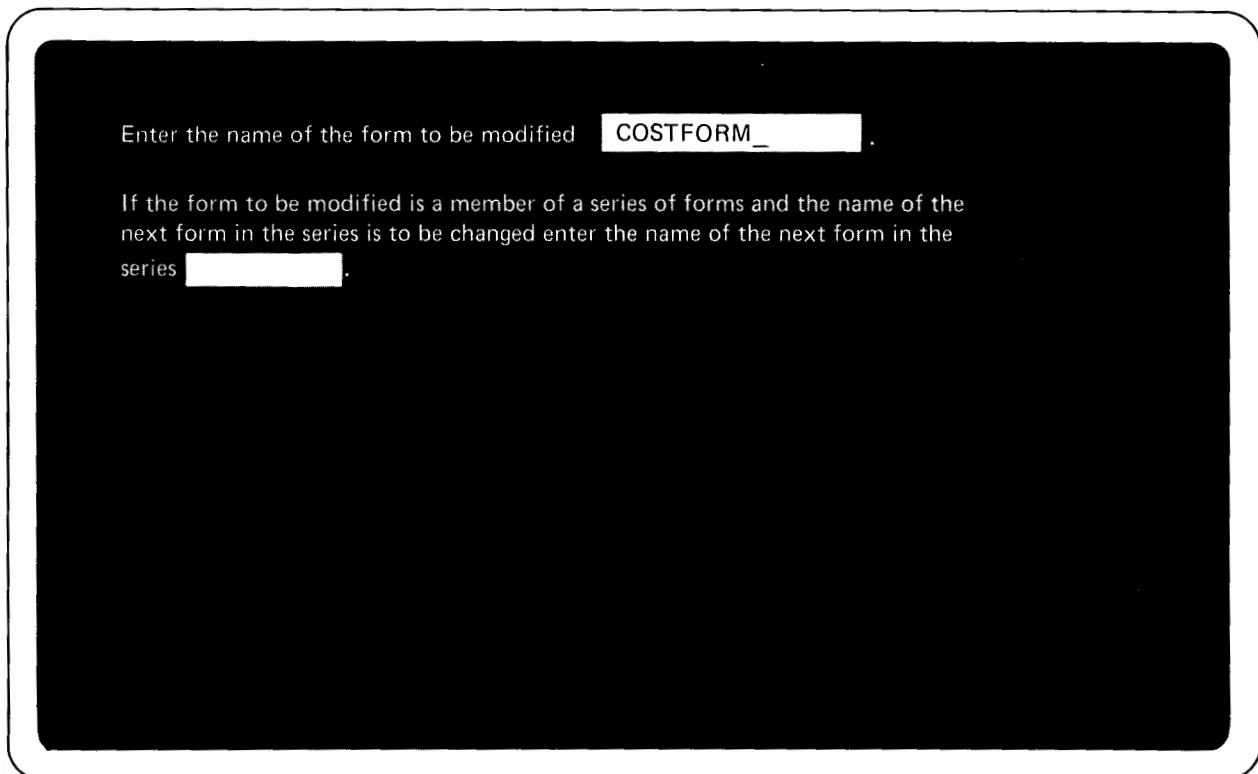
Transmit your responses to FORMAINTE by pressing ENTER. FORMAINTE then clears the screen and displays the form to be modified, allowing you to alter any entries you desire and to create new ones. Use the same methods you employed when first creating the form.

NOTE

The R_s notation, indicating the end of the form, is not displayed.

If you specified the name of a form that does not reside in the form file, the following message appears when you press ENTER and FORMAINTE waits for you to enter a correct form name:

FORM DOES NOT EXIST IN *filename*

A screenshot of the Form Modification Display. The background is black with white text. The first line says "Enter the name of the form to be modified" followed by a white rectangular input field containing the text "COSTFORM_". The second line is a multi-line instruction: "If the form to be modified is a member of a series of forms and the name of the next form in the series is to be changed enter the name of the next form in the series" followed by another white rectangular input field.

Enter the name of the form to be modified

If the form to be modified is a member of a series of forms and the name of the next form in the series is to be changed enter the name of the next form in the series

Figure 3-9. Form Modification Display

When you have made all necessary changes to the form, press ENTER. If the form contains unprotected fields, the Editing Specification Display (figure 3-7) appears, requiring you to re-specify all editing desired.

NOTE

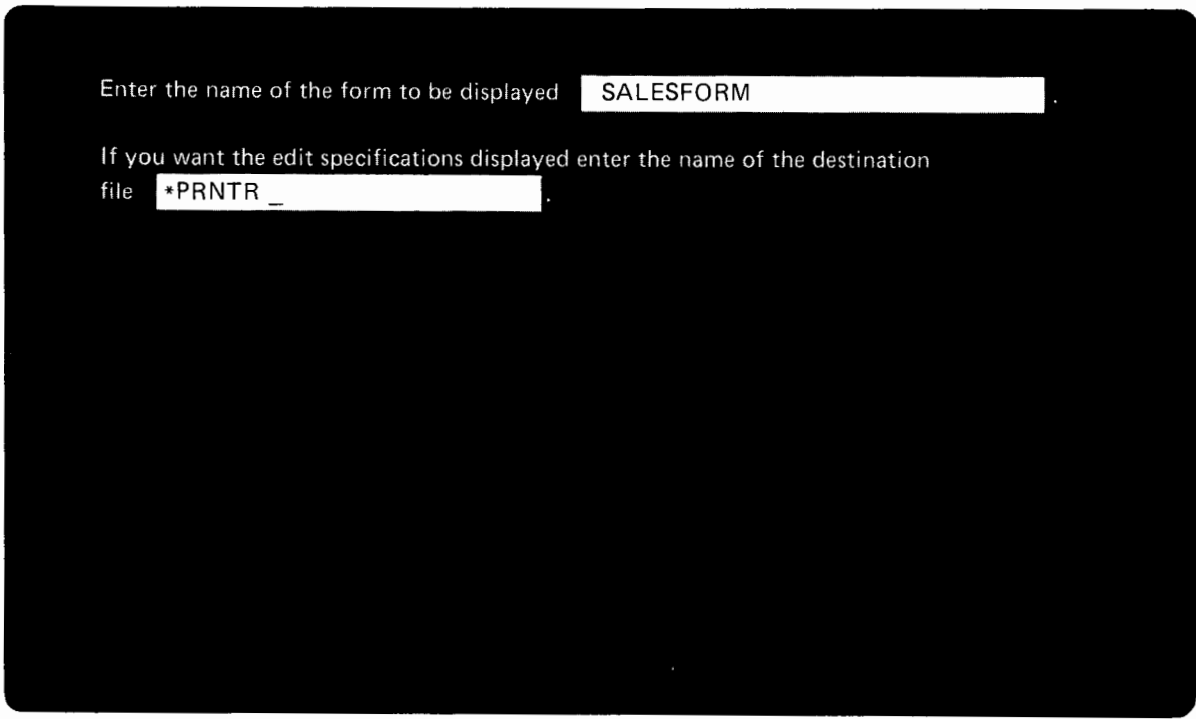
In addition to entering new or altered editing specifications, you must also *re-enter* any old, unchanged specifications that you wish to retain for the form.

To restore the form and editing specifications to the form file and return to the Function Selection Display, press ENTER.

DISPLAYING A FORM

You can display any form in the form file on your terminal. Alternatively, you can copy this form, plus any editing specifications for the form and other descriptive information, to an output device other than the terminal. To do this, enter X at the DISPLAY AN EXISTING FORM prompt and press ENTER. In response, FORMAINT presents the *Form Listing Selection Display* (figure 3-10). At the first prompt, enter the name of the form you wish displayed. If you also want to list the descriptive information and editing specifications, specify (at the second prompt) the name of another file/device to which this data can be transmitted; your terminal cannot be used for this purpose because it does not include enough memory to hold all of the information to be transmitted. The output file/device is typically a line printer, but other devices such as a disc or magnetic tape unit can be used. If you want to use a device other than a disc for the output, you must specify its file and device class names in an MPE :FILE command prior to running FORMAINT, as follows:

```
:FILE PRNTR;DEV= LP      File name of output file.
:RUN FORMAINT.PUB.SYS    Device class name of output device (a printer).
```



The screenshot shows a terminal window with a black background and white text. The first prompt is "Enter the name of the form to be displayed" followed by a text box containing "SALESFORM". The second prompt is "If you want the edit specifications displayed enter the name of the destination file" followed by a text box containing "*PRNTR _".

Figure 3-10. Form Listing Selection Display

When you enter the name of a non-disc file in the Form Listing Selection Display, you must also precede that name with an asterisk to indicate a back-reference to the :FILE command, as shown in figure 3-10. See *MPE Commands Reference Manual* for a discussion of back-referencing files.

Press ENTER to transmit the entries in the Form Listing Selection Display to FORMAIN.T.

You can request the display of *all* forms in the form file by entering a commercial-at sign (@) in response to the first prompt in the Form Listing Selection Display. If the listing device is your terminal, press ENTER after each form appears to display the next form, or enter f8 (HP 2644) or f8^c (HP 2640) to terminate the display. If the listing device is another device, all forms are listed in sequence automatically; you cannot terminate this operation prematurely.

The display output is written as 120-byte ASCII records. If, however, you entered the name of a form that does not exist in the form file, FORMAIN.T presents the following message and waits for you to enter a new form name:

FORM DOES NOT EXIST IN *filename*

If the output file for the form listing cannot be accessed, FORMAIN.T prints the following message and waits for you to enter a new file name. In this message, *nnnn* is the error code returned by the FCHECK intrinsic as discussed in *MPE Intrinsics Reference Manual*, and Appendix G of this manual.

LIST FILE CANNOT BE ACCESSED, ERROR CODE=*nnnn*

If FORMAIN.T displayed the form on your screen, press ENTER to clear the screen and return the Function Selection Display. If the form was written to another file, FORMAIN.T returns the Function Selection Display automatically. An output listing for the form SALESFORM, transmitted to a line printer, appears in figure 3-11.

DELETING A FORM

To delete a form from a form file, enter X at the DELETE AN EXISTING FORM prompt in the Function Selection Display and press ENTER. FORMAIN.T presents the *Form Deletion Display* (figure 3-12). Enter the form name at the prompt and press ENTER. The form is removed from the system immediately. If, however, you entered the name of a form that does not exist in the form file, FORMAIN.T prints the following message and waits for you to enter a correct name:

FORM DOES NOT EXIST IN *filename*

When the form is deleted, FORMAIN.T returns the Function Selection Display to the screen.

DELETING A FORM FILE

To delete an entire form file from the system, enter X at the DELETE THE FORM FILE prompt in the Function Selection Display and press ENTER. Because this action is irrevocable, FORMAIN.T outputs a *Form File Deletion Verification* prompt (figure 3-13) to ensure that you really want this file removed. To delete the file, enter YES; to keep it, enter NO. In either case, FORMAIN.T returns the Function Selection Display to the screen.

DESCRIPTIVE INFORMATION

FORM NAME IS SALESFORM CREATED 5/10/76 17:00
THIS FORM CONTAINS 35 INPUT FIELDS, TOTAL INPUT LENGTH IS 423 BYTES.
THERE ARE 23 EDITS SPECIFIED.

FORM

SALES TRANSACTION			
DATE: / /	TRANSACTION NO.		
CUSTOMER ID:	BILLED TO:		
SALESMAN ID:			
PRODUCT NO.	DESCRIPTION	UNIT PRICE	QUANTITY
		.	
		.	
		.	
		.	
		.	

Figure 3-11. Form Listing Output

TERMINATING FORMAINT

To terminate FORMAINT, closing all files accessed by it, enter X at the EXIT FORMAINT prompt in the Function Selection Display, and press ENTER. When FORMAINT terminates, MPE resumes control, printing the message shown below and prompting you for a new command:

END OF PROGRAM

: ← *prompt for new MPE command.*

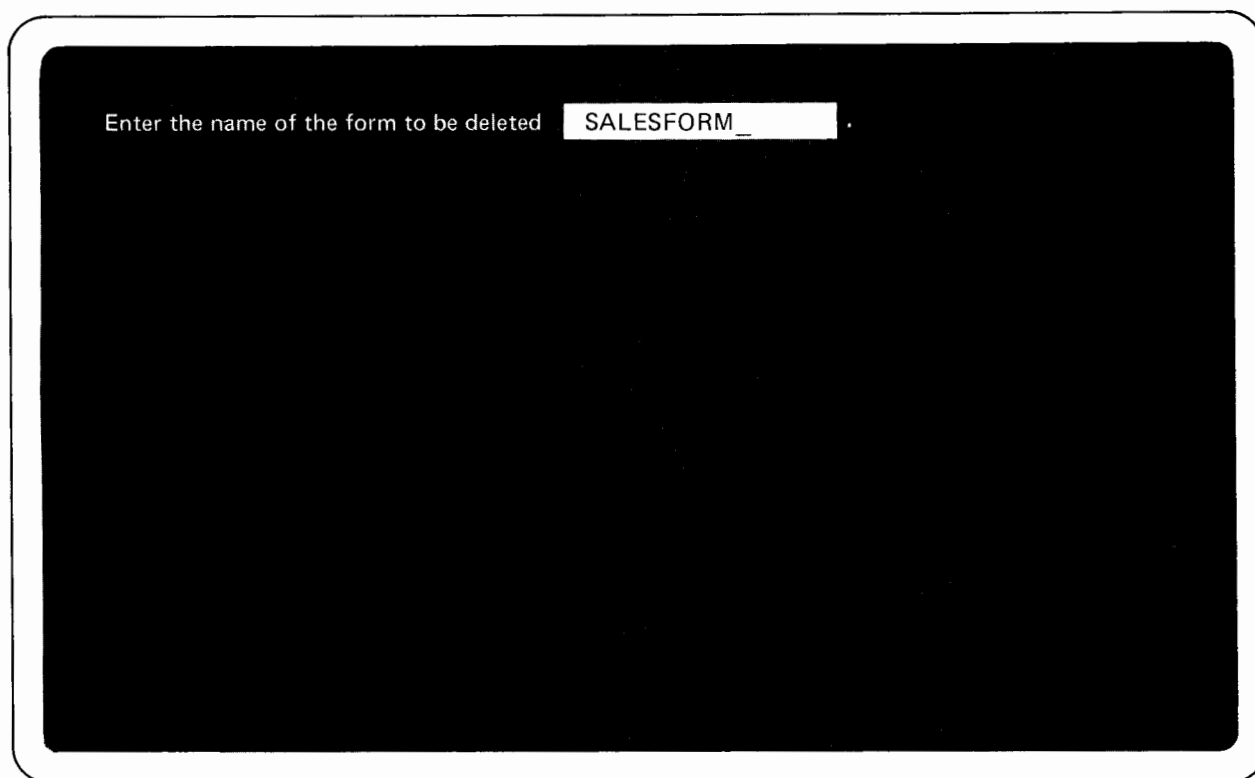
NOTE

After terminating FORMAINT but before entering a new MPE command, remove the terminal from block mode by releasing the BLOCK MODE key.

EDITING SPECIFICATIONS

FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
3	11	1	2	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNRANGE			0	0	0	01 12
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
3	14	3	2	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNRANGE			0	0	0	01 31
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
3	17	5	2	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNRANGE			0	0	0	0 70 79
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
3	59	7	6	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNUMRCEDIT			0	0	0	0
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
5	18	13	6	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNUMRCEDIT			0	0	0	0
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
5	54	19	23	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
6	54	42	23	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
7	18	65	6	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA
CNUMRCEDIT			0	0	0	0
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
7	54	71	23	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
8	54	94	23	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
14	5	117	4	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
14	18	121	39	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
14	18	160	41	0		
FIELD LOCATION			FIELD	NUMBER		
ROW	COL	IN INPUT	LENGTH	OF EDITS		
14	60	201	4	1		
PROCEDURE NAME			FLAGS TEST	SET SAME	OPPOSITE	PROCEDURE DATA

Figure 3-11. Form Listing Output (Continued)

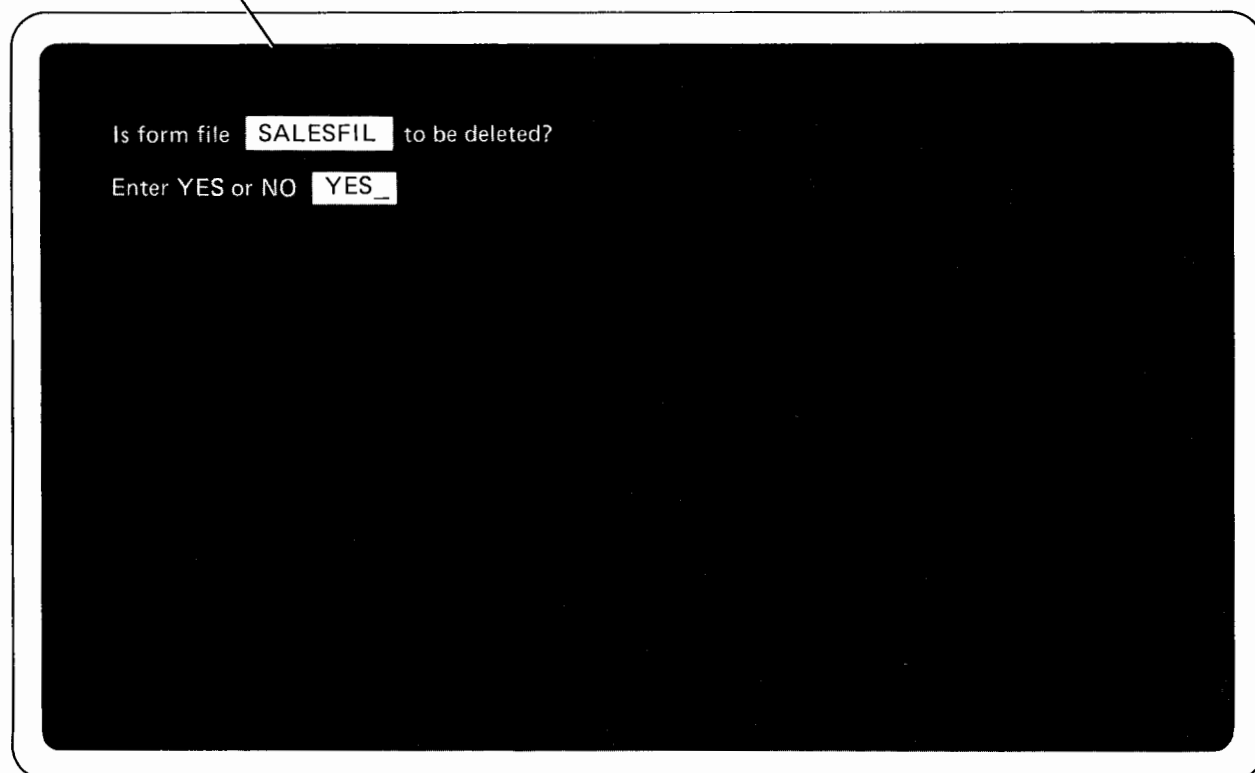


Enter the name of the form to be deleted

This is a screenshot of a terminal window with a black background and white text. The text prompts the user to enter the name of the form to be deleted, and the input field contains 'SALESFORM_'.

Figure 3-12. Form Deletion Display

*Name of file specified in
function selection display.*



Is form file to be deleted?
Enter YES or NO

This is a screenshot of a terminal window with a black background and white text. It asks for confirmation to delete the file 'SALESFIL'. The input field for the response contains 'YES_'. A line from the text above points to the 'SALESFIL' input field.

Figure 3-13. Form File Deletion Verification Display

USING DEL PROCEDURES

SECTION

IV

Once you have entered your form into the system, you are ready to write the application program that will access the form and interface with the user at the terminal. To do this, your program must call various procedures supplied by DEL. (You may also optionally supply additional specialized procedures that you write yourself for your DEL application. These procedures need not be written in the same language as your main program. In fact, you may write them in languages other than COBOL, FORTRAN, SPL and BASIC, as long as your program provides for any peculiarities involved in calling procedures in such languages.)

The DEL procedures fall into four basic functional categories, and are discussed in this order in Sections V through VIII:

- *Form access procedures* (Section V) that access the form file, including:

OPENFORM to open a form file.

FINDFORM to locate a form in the form file.

GETFORM to move a form into a program buffer (work area) called the form buffer.

NEXTEDIT to move the next edit specification into a program buffer called the edit buffer.

CLOSEFORM to close the form file.

- *Terminal access procedures* (Section VI) that access the terminal as a file, including:

OPENTERM to open the terminal as a file.

WRITETERM to write output from a program buffer to the terminal. (This output is typically the form moved into the form buffer by *GETFORM*.)

READTERM to read input from the terminal into a program buffer called the *data buffer*. (This input is the data entered by the person running the application program; at your option, the data buffer may occupy the same area of storage as the form buffer.)

TERMSTATUS to request status information about the terminal.

CLOSETERM to close the terminal file.

- *Editing procedures* (Section VII) that validate the general contents of input fields read from the terminal:

ALPHAEDIT to verify that the input field contains alphabetic characters (the letters A through Z) only.

ALPHAFILL to verify that the input field contains alphabetic characters only, with the last alphabetic character optionally followed to the right by blanks (spaces); no blanks, however, can be embedded *within* the alphabetic character string.

ANEDIT to verify that the input field contains alphanumeric characters (the letters A through Z), the digits 0 through 9, or spaces.

NUMRCEDIT to verify that the input field contains numeric data (the digits 0 through 9) only.

ZEROFILL to verify that the input field contains numeric data only, with the numeric string optionally preceded and/or followed by blanks. *ZEROFILL* also right-justifies this field and fills it to the left with zeros. No blanks, however, can appear *within* the numeric string.

NRANGE to ensure that the input field contains numeric data that falls within a specified range. The numeric string may be optionally preceded and/or followed by blanks, but may not *contain* blanks. This procedure, like *ZEROFILL*, right-justifies this field and fills it to the left with zeros.

M11CREATE to ensure that the input field contains numeric data only, generate a modulo-11 check digit, and insert this digit as the right-most digit of the input field.

M11VERIFY to ensure that the input field contains numeric data only, generate a modulo-11 check digit, and compare this with the right-most digit of the input field.

- *High-level procedures* (Section VIII) that combine some of the above procedures:

SHOWFORM to combine the *FINDFORM*, *GETFORM*, *WRITETERM*, and *READTERM* operations noted above in sequence; in essence, to locate and access a form, display the form on terminal screen, and read the user's input from terminal.

EDITFIELD to combine the *NEXTEDIT* and editing-procedure operations; effectively, to obtain the next editing procedure and then execute that procedure.

HOW PROCEDURES ARE USED

In your application program, the order in which you use the procedures depends partly on the application and partly upon your own preference. Typically, however, a program:

1. Initializes the required data areas and then opens the form file (*OPENFORM*) and terminal file (*OPENTERM*).
2. Locates the form (*FINDFORM*) and moves it into the form buffer (*GETFORM*).
3. Displays the form on the terminal (*WRITETERM*).
4. Reads the data that the user enters on the displayed form into the data buffer (*READTERM*).
5. Moves an editing specification into the edit buffer (*NEXTEDIT*).
6. Executes the editing procedure (*ALPHAEDIT*, *NUMRCEDIT*, and so forth).
7. Repeats steps 5 and 6, as applicable to the current input field.
8. Writes the user's data from the data buffer to a data file, using statements in the host program language. (The host language is that in which the application program is written.)

9. Chains to another form, if applicable.
10. Closes the form file (CLOSEFORM) and terminal file (CLOSETERM) when through.

After executing each procedure, the application program normally checks the status of the procedure to determine whether the procedure was executed successfully. (The status is reported in a special status *word*, discussed below.) The typical order of procedure calls is summarized in figure 4-1, where it is also compared with a sequence of calls that combines some operations by invoking SHOWFORM and EDITFIELD.

COMMON PROCEDURE CHARACTERISTICS

All DEL procedures can be accessed from programs written in COBOL, FORTRAN, BASIC, and SPL. The same name is used for each procedure in FORTRAN, BASIC, and SPL programs. For instance, in all three languages, you invoke the procedure that opens your form file by using the procedure name *OPENFORM*. In COBOL programs, however, you must prefix the standard procedure name with the letter C; this denotes an alternate entry-point for the procedure that must be used by COBOL programs because these programs pass parameters somewhat differently than FORTRAN, BASIC, and SPL programs. Thus, to open the form file in a COBOL program, you would invoke the OPENFORM procedure by using the name *COPENFORM*.

All procedures require at least one parameter: the name of the *DEL communication area (commarea)*, a global storage area in memory used by DEL to maintain information about the form file and terminal that the application program is using. This information includes the file numbers that identify these files to the MPE File Management System, information about the currently-used form, data regarding the editing procedures for the form, and various pointers to areas on the form and other data. This area occupies 128 contiguous words of memory, the first of which is the status word. When each procedure terminates, it sets the status word with a value that indicates whether the procedure succeeded or failed. For FORTRAN, BASIC, and SPL programs, this value is designated as type INTEGER. For COBOL programs, it is designated as USAGE COMP. When a procedure returns the value *zero* to this word, this indicates successful (normal) completion. When it returns any other value, an abnormal condition occurred; the specific meaning of a value depends both upon the type of abnormality encountered and the procedure that returned the value.

The remaining 127 words in the communication area are used by DEL. The format of the entire communication area is summarized in Appendix A.

Various DEL procedures may refer to any of the parameters defined in table 4-1 and noted throughout this manual. Those parameters defined as type *CHARACTER* are assumed to reference data whose first byte (character) is located at the beginning of a memory word. When you define a data item or area referenced by one of these parameters in your application program, make certain that you assign it the characteristics specified for it in table 4-1. For instance, always assign the form buffer (*formbuffer* parameter) a minimum length of 64 bytes, and specify it as a data item of type CHARACTER. For further information, see the program example in Section IX.

PROGRAM WITHOUT COMBINED OPERATIONS	PROGRAM WITH COMBINED OPERATIONS
<pre> . . . "OPENFORM" (CHECK STATUS) "OPENTERM" (CHECK STATUS) . . . (A) "FINDFORM" (CHECK STATUS) "GETFORM" (CHECK STATUS) (B) "WRITETERM" (CHECK STATUS) "READTERM" (CHECK STATUS) . . . "NEXTEDIT" (CHECK STATUS) "NUMRCEDIT" (CHECK STATUS) . . . "NEXTEDIT" (CHECK STATUS) "ALPHAEDIT" (CHECK STATUS) . . . SAME FORM? GO TO (B) NEW FORM? GO TO (A) "CLOSEFORM" (CHECK STATUS) "CLOSETERM" (CHECK STATUS) . . . </pre>	<pre> . . . "OPENFORM" (CHECK STATUS) "OPENTERM" (CHECK STATUS) . . . "SHOWFORM" (CHECK STATUS) . . . "EDITFIELD" (CHECK STATUS) . . . "EDITFIELD" (CHECKSTATUS) . . . "CLOSEFORM" (CHECK STATUS) "CLOSETERM" (CHECK STATUS) . . . </pre>

Figure 4-1. Procedure Calls in Application Programs

Table 4-1. Summary of Procedure Parameters

PARAMETER	MEANING	DATA ITEM TYPE	AREA/VARIABLE LENGTH
<i>commarea</i>	Name of DEL communication area used for your program. This is a 128-word area whose first entry is <i>status word</i> that can be interrogated by your program to determine if last procedure called was executed successfully. The status word is a 16-bit numeric quantity that must be declared as usage COMP for COBOL programs and type INTEGER for all other programs. The remaining 127-words are used for global storage by DEL.	INTEGER	128 words exactly
<i>databuffer</i>	Name of program buffer used for input of user data from terminal.	CHARACTER	Any length depending upon number of input characters expected.
<i>datalength</i>	Name of variable used to specify length of <i>databuffer</i> ; DEL also returns length of data actually read to this variable.	NUMERIC	2 bytes (1 word) exactly.
<i>editdef</i>	Name of program buffer to hold specifications for edit procedures.	CHARACTER	72 bytes (characters) exactly.
<i>fbufferlength</i>	Name of variable containing length of form buffer; DEL also returns actual length of form and its allied information to this buffer.	NUMERIC	2 bytes (1 word) exactly.
<i>formbuffer</i>	Name of program buffer to which form is moved from form file, and from which form is written to terminal.	CHARACTER	64 bytes minimum.
<i>formfile</i>	Name of area containing MPE formal file designator of current form file.	CHARACTER	35 bytes (characters) maximum.
<i>formlength</i>	Name of variable to which length of current form is returned.	NUMERIC	2 bytes (1 word) exactly.
<i>formname</i>	Name of area containing identity of current form.	CHARACTER	16 bytes (characters) exactly.
<i>nextform</i>	Name of area to which name of next form in a chained series is returned.	CHARACTER	16 bytes (characters) exactly.
<i>statbuffer</i>	Name of buffer to which status of terminal is returned.	CHARACTER	28 bytes minimum.
<i>termfile</i>	Name of area containing MPE formal file designator of terminal file.	CHARACTER	8 bytes (characters) maximum.

ACCESSING FORMS FROM PROGRAMS

SECTION

V

Using the procedures discussed in this section, your program can obtain read-only access to the form file. One of these procedures, GETFORM, enables your program to copy a form residing in the form file into a program buffer. Then, by calling the WRITETERM procedure, discussed in Section VI, the program can display the form on the terminal screen. After the user at the terminal enters data on the form, your program can read this data into a program buffer by calling the READTERM procedure, also discussed in Section VI. This buffer can be the same one used by the GETFORM procedure, or it can be a different one. (Subsequently, the program may transfer the data to a data file, using its own host-language statements.) These steps, which generally form the heart of most DEL application programs, are illustrated in figure 5-1. An explicit example of these steps appears in the sample program in Section IX.



OPENING A FORM FILE

Before your application program can select a form, it must initiate access to (open) the form file that contains the form. To open the form file, call the OPENFORM procedure. If successful, this procedure returns the value zero to the status word in the DEL communication area. If the file is not a form file created by FORMINT, this procedure sets the status word to minus one. If the file cannot be opened, the procedure returns an FCHECK error code (a value greater than zero) to the status word. See the discussion of the FCHECK intrinsic in the *MPE Ininsics Reference Manual* or the FCHECK error code summary in Appendix G of this manual for the meaning of this code.

The calling sequence for the OPENFORM procedure appears below, as entered in the COBOL, FORTRAN, SPL, and BASIC languages. In this sequence and those for all other DEL procedures, entries in UPPER-CASE CHARACTERS must be written exactly as shown. Entries in *lower-case*

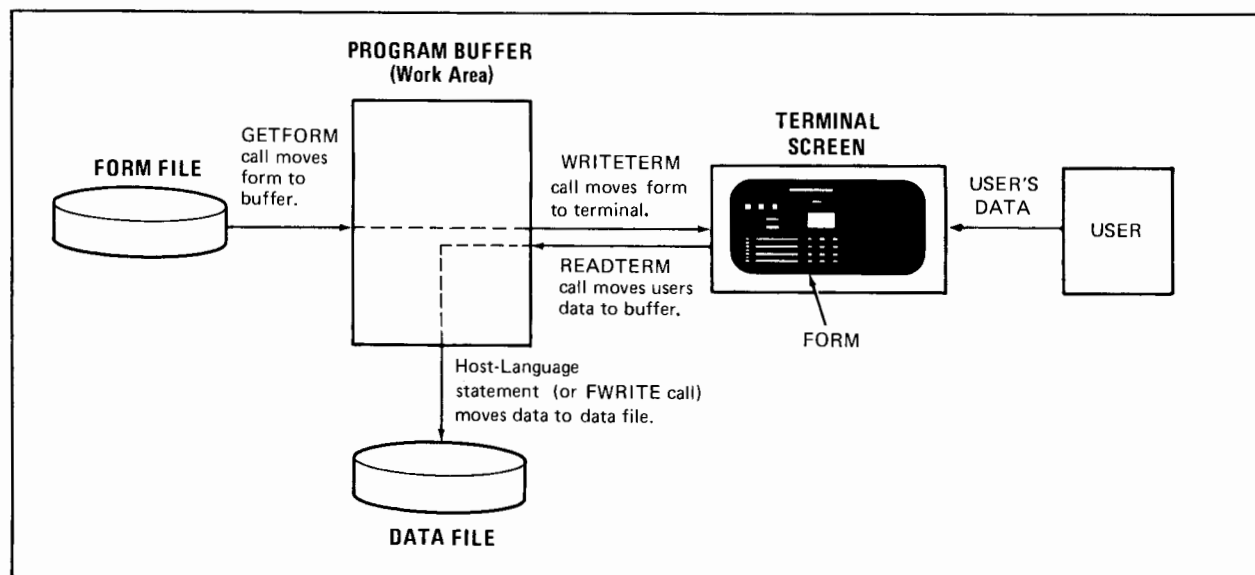


Figure 5-1. Application Program Procedures in Action

italics are user-defined variables that you must supply. All parameters shown are required and cannot be omitted. Unless the host language allows embedded blanks in procedure calls as a generality, do not include them in these DEL procedure calls. Examples of calls also appear below.

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "COPENFORM" USING <i>commarea formfile</i> . Example: CALL "COPENFORM" USING MYCOMM MYFORMFL.
FORTRAN	CALL OPENFORM(<i>commarea,formfile</i>) Example: CALL OPENFORM(MYCOMM,MYFORMFL)
SPL	OPENFORM(<i>commarea,formfile</i>); Example: OPENFORM(MYCOMM,MYFORMFL);
BASIC	CALL OPENFORM(<i>c(1),f\$</i>) Example: CALL OPENFORM(C1(1),F\$)

The parameters in the OPENFORM calling sequence indicate the following:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. This area is a 128-word integer data item whose first entry is the <i>status word</i> that can be interrogated by your program to determine if this and other procedures executed successfully. The status word is a 16-bit numeric quantity that must be declared as usage COMP for COBOL programs and type INTEGER for all other programs. The remaining 127 words of the communications area are used for global storage by DEL.
<i>formfile</i> <i>f\$</i>	The name of the area containing the MPE formal file designator that identifies the form file. This is a character data item with a maximum length of 35 bytes.

Before using the above parameters in procedure calls, you should define them within your program and initialize the areas they reference. The DEL communication area, for instance, cannot be used by any procedure unless it is defined as an integer data item with the status word declared as usage COMP for COBOL programs and type INTEGER for all others. The area referenced by *formfile* must be defined as a character data item, and should already contain the form file designator when you reference it in any procedure call. This kind of rule applies to *all* parameters referenced by all procedures discussed in this manual.

NOTE

You must call OPENFORM before you can call any other procedure described in this section.

LOCATING A FORM IN A FORM FILE

Before your program can perform any operation on a form, it must locate that form in the form file. To do this, the program calls the FINDFORM procedure, which places current information about the form in the DEL communication area for use by subsequent procedures, clears any edit flags by re-setting them to zero, and returns the length of the form and the name of the next (chained) form, if any, to areas established by your program. If these operations are successful, the procedure sets the status word to zero; if the form is not stored in the form file, the procedure sets the status word to minus 1; if the form file cannot be read, the procedure sets the status word to the appropriate FCHECK error code.

NOTE

Before your program can issue any calls to FINDFORM, it must have issued an OPENFORM call to open the form file.

The calling sequence for FINDFORM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CFINDFORM" USING <i>commarea formname formlength nextform</i> . Example: CALL "CFINDFORM" USING MYCOMM MYFORM LENF NEXTF.
FORTRAN	CALL FINDFORM(<i>commarea,formname,formlength,nextform</i>) Example: CALL FINDFORM(MYCOMM,MYFORM,LENF,NEXTF)
SPL	FINDFORM(<i>commarea,formname,formlength,nextform</i>); Example: FINDFORM(MYCOMM,MYFORM,LENF,NEXTF);
BASIC	CALL FINDFORM(<i>c(1),f1\$,l,f2\$</i>) Example: FINDFORM(C1(1),F1\$,L,F2\$)

The parameters in this calling sequence indicate:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
<i>formname</i> <i>f1\$</i>	The name of the area that contains the name of your form. This area is a CHARACTER data item that must be exactly 16 bytes (characters) long.
<i>formlength</i> <i>l</i>	The name of a variable to which FINDFORM returns the length of the form accessed. This is a one-word NUMERIC data item. This item must be declared usage COMP in COBOL programs.
<i>nextform</i> <i>f2\$</i>	The name of the area to which FINDFORM returns the name of the next form in the chained series, if the current form belongs to such a series. This is a CHARACTER data item that must be exactly 16 bytes long.

MOVING A FORM TO A PROGRAM BUFFER

When your program is ready to move a form from the current form file into a buffer, it calls the GETFORM procedure. This procedure also transfers terminal control characters to the buffer; these characters are used to clear the terminal screen, lock the keyboard, set format mode, and perform other functions needed when your program later displays the form on the terminal screen. (See Section VI.) If the program buffer is too small to hold the entire form, GETFORM updates current information in the communication area so that your program can retrieve the remainder of the form by issuing a second GETFORM call. If the GETFORM operation is successful, this procedure sets the status word to zero. If the form file cannot be read, the procedure sets the status word to the appropriate FCHECK error code.

NOTE

Before your program can call GETFORM, it must have called OPENFORM to open the form file and FINDFORM to locate the form.

The calling sequence for GETFORM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CGETFORM" USING <i>commarea formbuffer fbufferlength</i> . Example: CALL "CGETFORM" USING MYCOMM MYFBUF LENB.
FORTRAN	CALL GETFORM(<i>commarea,formbuffer,fbufferlength</i>) Example: CALL GETFORM(MYCOMM,MYFBUF,LENB)
SPL	GETFORM(<i>commarea,formbuffer,fbufferlength</i>); Example: GETFORM(MYCOMM,MYFBUF,LENB);
BASIC	CALL GETFORM(<i>c(1),b\$,l</i>) Example: CALL GETFORM(C1(1),B0\$,L0)

The parameters in this calling sequence are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
<i>formbuffer</i> <i>b\$</i>	The name of the program buffer to which GETFORM copies the form and its allied information. This must be a CHARACTER data item with a minimum length of 64 bytes.
<i>fbufferlength</i> <i>l</i>	The name of a variable that contains the length of your program buffer, and to which GETFORM returns the actual length of the form and its allied information. This must be a one-word NUMERIC data item.

NOTE

The *fbufferlength* parameter for this procedure should typically specify a different area than the *formlength* parameter for FINDFORM, to avoid overwriting the information returned by FINDFORM.

RETRIEVING NEXT EDIT SPECIFICATIONS

After your program has moved the form into its buffer (GETFORM procedure), displayed it upon the terminal screen (WRITETERM procedure, Section VI), and read the user's data (READTERM procedure, also Section VI), the program should then apply any editing specifications necessary to the data read. Before the program can call the procedures for these specifications, however, it must move each of them in sequence into a program buffer. This is done through the NEXTEDIT procedure, which also updates the appropriate editing pointers in the DEL communication area. For example, if your program is to use two editing procedures in sequence, ANEDIT and NRANGE, it would:

1. Call NEXTEDIT to move the ANEDIT specifications into the buffer.
2. Call ANEDIT to perform an alphanumeric edit check.
3. Call NEXTEDIT to move the NRANGE specification into the buffer.
4. Call NRANGE to perform a numeric range check on the data.

If the last edit specification has already been accessed when NEXTEDIT is called, the status word is set to minus one, the program buffer is unchanged, and the communication area is reset to indicate the first edit specification. If the next editing specification cannot be read from the form file, NEXTEDIT sets the status word to the appropriate FCHECK error code.

NOTE

Before your program can call NEXTEDIT, it must have called FINDFORM to locate the form and its editing specifications.

The calling sequence for NEXTEDIT is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CNEXTEDIT" USING <i>commarea editdef</i> . Example: CALL "CNEXTEDIT" USING MYCOMM MYEDEF.
FORTRAN	CALL NEXTEDIT(<i>commarea,editdef</i>) Example: CALL NEXTEDIT(MYCOMM,MYEDEF)
SPL	NEXTEDIT(<i>commarea,editdef</i>); Example: NEXTEDIT(MYCOMM,MYEDEF);
BASIC	CALL NEXTEDIT(<i>c(1),e\$</i>) Example: CALL NEXTEDIT(C1(1),E\$)

The parameters in the calling sequence are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
<i>editdef</i> <i>e\$</i>	The name of the program buffer to which NEXTEDIT copies the next editing specification. This must be a CHARACTER data item with a minimum length of 72 bytes.

NOTE

The *editdef* parameter for this procedure should always specify a different area from the *databuffer* parameter of the READTERM procedure to avoid overwriting the information returned by READTERM. READTERM reads the user's data (refer to Section VI).

CLOSING A FORM FILE

When your program has finished using the form file, it closes it by calling the CLOSEFORM procedure. If this operation fails, CLOSEFORM sets the status word to the appropriate FCHECK error code. The CLOSEFORM calling sequence is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CCLOSEFORM" USING <i>commarea</i> . Example: CALL "CCLOSEFORM" USING MYCOMM.
FORTRAN	CALL CLOSEFORM(<i>commarea</i>) Example: CALL CLOSEFORM(MYCOMM)
SPL	CLOSEFORM(<i>commarea</i>); Example: CLOSEFORM(MYCOMM);
BASIC	CALL CLOSEFORM(<i>c</i> (1)) Example: CALL CLOSEFORM(C1(1))

The only parameter in this calling sequence is:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure definition.)

ACCESSING TERMINALS FROM PROGRAMS

SECTION

VI

With the procedures discussed in this section, your program can obtain read/write access to the terminal as an MPE file. Thus, your program can call the WRITETERM procedure to copy a form from its program buffer (*formbuffer*) onto the terminal screen. The program can also call the READTERM procedure to move the data that the user at the terminal enters into a program buffer (*databuffer*). The *formbuffer* and *databuffer* can occupy a common area of memory, or they can be defined as two distinct areas. Other procedures are also available for use in support of WRITETERM and READTERM.

OPENING A TERMINAL FILE

Just as your application program must open the form file before it can select a form (Section V), it must open the user's terminal as an MPE file before it can access that terminal. To open the terminal/file and verify that the terminal is either an HP 2640 or 2644 terminal, the program calls the OPENTERM procedure. If successful, this procedure returns the value zero to the status word in the DEL communication area. If the terminal is not a 2640 or 2644, the procedure sets the status word to minus one. If the terminal/file cannot be opened, the procedure returns an FCHECK error code (a value greater than zero) to the status word. The calling sequence for OPENTERM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "COPENTERM" USING <i>commarea termfile</i> . Example: CALL "COPENTERM" USING MYCOMM MYTERMFL.
FORTRAN	CALL OPENTERM(<i>commarea,termfile</i>) Example: CALL OPENTERM(MYCOMM,MYTERMFL)
SPL	OPENTERM(<i>commarea,termfile</i>); Example: OPENTERM(MYCOMM,MYTERMFL);
BASIC	CALL OPENTERM(<i>c(1)</i>), <i>t\$</i> Example: CALL OPENTERM(C1(1),T\$

The parameters for OPENTERM are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
<i>termfile</i> <i>t\$</i>	The name of the area containing the MPE formal file designator that identifies the terminal file. This is a character data item with a maximum length of 8 bytes.

NOTE

You must call OPENTERM before you can call any other terminal access procedure described in this section.

WRITING OUTPUT TO A TERMINAL

When your program is ready to display a form onto a terminal, it calls the WRITETERM procedure. This procedure transfers the entire contents of the specified form buffer to the user's terminal. If the procedure succeeds, it sets the DEL status word to zero. If the write operation fails, the procedure sets the status word to the appropriate FCHECK error code.

NOTE

Before your program can call WRITETERM, it must have opened the terminal as a file by calling OPENTERM.

The calling sequence for WRITETERM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CWRITETERM" USING <i>commarea formbuffer fbufferlength</i> . Example: CALL "CWRITETERM" USING MYCOMM MYFBUF LENB.
FORTRAN	CALL WRITETERM(<i>commarea,formbuffer,fbufferlength</i>) Example: CALL WRITETERM(MYCOMM,MYFBUF,LENB)
SPL	WRITETERM(<i>commarea,formbuffer,fbufferlength</i>); Example: WRITETERM(MYCOMM,MYFBUF,LENB);
BASIC	CALL WRITETERM(<i>c(1),b\$,l</i>) Example: CALL WRITETERM(C1(1),B0\$,L0)

The parameters in this calling sequence are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
<i>formbuffer</i> <i>b\$</i>	The name of the program buffer from which WRITETERM copies the form. This must be a CHARACTER data item with a minimum length of 64 bytes.
<i>fbufferlength</i> <i>l</i>	The name of a variable that contains the length of the data to be transferred from the buffer. This must be a one-word NUMERIC data item.

READING INPUT FROM A TERMINAL

When your program is ready to accept the data a user entered at the terminal, it calls the READTERM procedure. This procedure reads the data from all unprotected areas on the screen and copies it to the designated program buffer. This buffer may be the same one used by the GETFORM and WRITETERM procedures, or it may be another buffer. It must *not*, however, be the same buffer used by any NEXTEDIT procedure called to edit the input data — otherwise, the editing specifications

needed for this edit will overwrite the input data. The READTERM procedure also sets the one-word variable *datalength* to indicate the length of the data actually read from the terminal. If READTERM executes successfully, it sets the DEL status word to zero. If the procedure fails, it sets the status word to the appropriate FCHECK error code.

NOTE

Before your program can call READTERM, it must have called OPENTERM to open the terminal/file.

The calling sequence for READTERM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CREADTERM" USING <i>commarea databuffer datalength</i> . Example: CALL "CREADTERM" USING MYCOMM MYDBUF LENDB.
FORTRAN	CALL READTERM(<i>commarea,databuffer,datalength</i>) Example: CALL READTERM(MYCOMM,MYDBUF,LENDB)
SPL	READTERM(<i>commarea,databuffer,datalength</i>); Example: READTERM(MYCOMM,MYDBUF,LENDB);
BASIC	CALL READTERM(<i>c{1},b\$,l</i>) Example: CALL READTERM(C1{1},B1\$,L1)

The READTERM parameters are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
<i>databuffer</i> <i>b\$</i>	The name of the program buffer to receive the user's input from the terminal. This buffer can be any length, depending upon the number of characters expected in the data entered. It must be a CHARACTER type data item.
<i>datalength</i> <i>l</i>	The name of a variable specifying the actual length of the data buffer; when READTERM is executed, the procedure returns to this variable the actual length of the data read. This must be a one-word NUMERIC type variable.

REQUESTING TERMINAL STATUS

From time to time, a program might require information about the operating status of the terminal from which the user enters his data. For example, you might wish your program to periodically determine that the terminal is still in block mode while the user is entering his data. Other useful information might include terminal memory size, strapping options in effect, latching key status (latched/unlatched), input/output transfers pending, error conditions, or other data. (See the owner's manual for your terminal for further details on what particular information is maintained.) To obtain this information, the program calls the TERMSTATUS procedure, which writes the status information into a buffer defined by the program. For both the HP 2640 and 2644 terminals, the first six bytes of

the buffer receive the terminal memory size, reported in decimal digits; the remaining bytes receive the terminal error flag settings, reported as 0 (for OFF) and 1 (for ON). The HP 2640 maintains six error flags, while the HP 2644 maintains 22. Because the procedure returns the maximum number of bytes possible with either type of terminal, the program buffer should be at least 28 bytes long. If the status information is written to the buffer, the procedure sets the DEL status word to zero. If the status information is not written, it returns the appropriate FCHECK code to the DEL status word.

NOTE

Before your program can call the TERMSTATUS procedure, it must have called the OPENTERM procedure to open the terminal/file.

The TERMSTATUS calling sequence is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CTERMSTATUS" USING <i>commarea statbuffer</i> . Example: CALL "CTERMSTATUS" USING MYCOMM MYSTATUS.
FORTRAN	CALL TERMSTATUS(<i>commarea,statbuffer</i>) Example: CALL TERMSTATUS(MYCOMM,MYSTATUS)
SPL	TERMSTATUS(<i>commarea,statbuffer</i>); Example: TERMSTATUS(MYCOMM,MYSTATUS);
BASIC	CALL TERMSTATUS(<i>c(1),s\$</i>) Example: CALL TERMSTATUS(C1(1),S1\$)

The TERMSTATUS parameters are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
<i>statbuffer</i> <i>s\$</i>	Name of buffer to which terminal memory size and status-flag settings are returned. This must be a CHARACTER type data item at least 28 bytes long.

CLOSING A TERMINAL FILE

When your program has finished using the terminal file, it closes this file by calling the CLOSETERM procedure. If this procedure executes successfully, it sets the DEL status word to zero. If this procedure cannot close the file, it sets the status word to the appropriate FCHECK error code. The CLOSETERM calling sequence is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CCLOSETERM" USING <i>commarea</i> . Example: CALL "CCLOSETERM" USING MYCOMM.
FORTRAN	CALL CLOSETERM(<i>commarea</i>) Example: CALL CLOSETERM(MYCOMM)
SPL	CLOSETERM(<i>commarea</i>); Example: CLOSETERM(MYCOMM);
BASIC	CALL CLOSETERM(<i>c</i> (1)) Example: CALL CLOSETERM(C1(1))

The CLOSETERM parameters are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure definition in Section V.)

CALLING EDIT PROCEDURES

SECTION

VII



DEL provides eight procedures that perform editing functions on data entered at terminals. Edit procedures are called from your programs by first calling the NEXTEDIT procedure described in Section V, and then calling the desired edit function. After performing the edit function, a pass/fail indication is returned to the first word (status word) of the DEL communications area. If the input data *fails* the edit procedure that was called, the status word is set to *minus one*; if the input data *passes* the edit, the status word is set to *zero*.

Your program should read the status word to determine whether the data passed or failed the edit, and then take whatever appropriate action you desire. The DEL edit procedures do not interact with the terminal operator, since the system does not keep sufficient information about the current contents of the terminal memory to select an unoccupied area of the screen for displaying an error message. If your application requires error messages and interaction with the terminal operator, you must provide the additional programming to accomplish this.

The eight DEL edit procedures listed below are described in the following pages.

Alphabetic data field	(ALPHAEDIT)
Alphabetic space-filled field	(ALPHAFILL)
Alphanumeric data field	(ANEDIT)
Numeric data field	(NUMRCEDIT)
Numeric zero-filled field	(ZEROFILL)
Numeric range edit	(NRANGE)
Modulo 11 check digit create	(M11CREATE)
Modulo 11 check digit verify	(M11VERIFY)

ALPHABETIC DATA FIELD (ALPHAEDIT)

ALPHAEDIT checks the data in the input field to determine if all of the characters are alphabetic. The procedure scans the input field; if any character is not one of the letters A through Z, the data fails the edit, and the status word in the DEL communication area is set to minus one. If the data passes the edit, the status word is set to zero. No spaces, numbers or special characters are allowed.

The calling sequence for the ALPHAEDIT edit procedure appears below, as entered in the COBOL, FORTRAN, SPL, and BASIC languages. As described in Section V, entries in UPPER-CASE CHARACTERS must be written exactly as shown, and entries in *lower-case italics* are user-defined variables that you must supply. Examples of each calling procedure are included below.

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CALPHAEDIT" USING <i>commarea editdef databuffer</i> . Example: CALL "CALPHAEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ALPHAEDIT(<i>commarea,editdef,databuffer</i>) Example: CALL ALPHAEDIT(MYCOMM,MYEDEF,MYDBUFF)
SPL	ALPHAEDIT(<i>commarea,editdef,databuffer</i>); Example: ALPHAEDIT(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL ALPHAEDIT(<i>c(1),e\$,b\$</i>) Example: CALL ALPHAEDIT(C1(1),E\$,B1\$)

The parameters in the ALPHAEDIT calling sequence indicate the following:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area, a 128-word integer data item whose first entry is the status word that can be interrogated by your program to determine if this and other procedures executed successfully.
<i>editdef</i> <i>e\$</i>	The name of a CHARACTER-type data item containing the edit specifications as returned by NEXTEDIT. This item must be exactly 72 bytes (characters) long.
<i>databuffer</i> <i>b\$</i>	The name of a CHARACTER-type data item containing the data from the terminal as returned by READTERM. This item can be any length, depending on the number of input characters expected.

ALPHABETIC SPACE-FILLED FIELD (ALPHAFILL)

ALPHAFILL allows you to check the data in the input field to determine if the data consists of alphabetic characters *and* spaces to the right of the last alphabetic character. The procedure scans the input field; the characters must be one of the letters A through Z, *or* any number of spaces to the right of the last alphabetic character. If the input data fails the edit, the status word is set to minus one; if the data passes the edit, the status word is set to zero. No embedded spaces, numbers or special characters are allowed.

The calling sequence for the ALPHAFILL edit procedure appears below, as entered in the COBOL, FORTRAN, SPL, and BASIC languages.

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CALPHAFILL" USING <i>commarea editdef databuffer</i> . Example: CALL "CALPHAFILL" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ALPHAFILL(<i>commarea,editdef,databuffer</i>) Example: CALL ALPHAFILL(MYCOMM,MYEDEF,MYDBUFF)
SPL	ALPHAFILL(<i>commarea,editdef,databuffer</i>); Example: ALPHAFILL(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL ALPHAFILL(<i>c(1),e\$,b\$</i>) Example: CALL ALPHAFILL(C1(1),E\$,B1\$)

The parameters (*commarea*, *c1*, *editdef*, *e\$*, *databuffer*, *b\$*) as described for the ALPHAEDIT calling sequence are the same for all eight DEL edit procedures.

ALPHANUMERIC DATA FIELD (ANEDIT)

ANEDIT allows you to check the data in the input field to determine if all characters are alphabetic characters, numeric digits, or spaces. The procedure scans the input field; if any character is not one of the letters A through Z, *or* one of the digits 0 through 9, *or* a space, the status word is set to minus one. If the data passes the edit, the status word is set to zero. No special characters (except spaces) are allowed.

NOTE

This is the only DEL editing procedure that permits embedded spaces.

The calling sequence of the ANEDIT procedure appears below.

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CANEDIT" USING <i>commarea editdef databuffer</i> . Example: CALL "CANEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ANEDIT(<i>commarea,editdef,databuffer</i>) Example: CALL ANEDIT(MYCOMM,MYEDEF,MYDBUFF)
SPL	ANEDIT(<i>commarea,editdef,databuffer</i>); Example: ANEDIT(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL ANEDIT(<i>c(1),e\$,b\$</i>) Example: CALL ANEDIT(C1(1),E\$,B1\$)

NUMERIC DATA FIELD (NUMRCEDIT)

NUMRCEDIT allows you to check the data in the input field to determine if all of the characters are numeric digits. The procedure scans the field; if any character is not one of the digits 0 through 9, the status word is set to minus one. If the data passes the edit, the status word is set to zero. No spaces, alphabetic characters or special characters are allowed.

The calling sequence of the NUMRCEDIT edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CNUMRCEDIT" USING <i>commarea editdef databuffer</i> . Example: CALL "CNUMRCEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL NUMRCEDIT(<i>commarea,editdef,databuffer</i>) Example: CALL NUMRCEDIT(MYCOMM,MYEDEF,MYDBUFF)
SPL	NUMRCEDIT(<i>commarea,editdef,databuffer</i>); Example: NUMRCEDIT(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL NUMRCEDIT(<i>c(1),e\$,b\$</i>) Example: CALL NUMRCEDIT(C1(1),E\$,B1\$)

NUMERIC ZERO-FILLED FIELD (ZEROFILL)

ZEROFILL allows you to check the data in the input field to determine if the data consists of only numeric digits *and/or* spaces. The spaces can appear before and/or after the numeric digits, but the edit does not allow embedded spaces. As part of the ZEROFILL edit procedure all spaces are stripped from the numeric digits, the digits are right-justified, and the field is filled with zeros to the left of the digits. The procedure then scans the input field; if any character is not a digit, 0 through 9, the data is considered failed, and the status word is set to minus one. If the data passes the edit, the DEL status word is set to zero. No embedded spaces, alphabetic characters or special characters are allowed.

The calling sequence of the ZEROFILL edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CZEROFILL" USING <i>commarea editdef databuffer</i> . Example: CALL "CZEROFILL" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ZEROFILL(<i>commarea,editdef,databuffer</i>) Example: CALL ZEROFILL(MYCOMM,MYEDEF,MYDBUFF)
SPL	ZEROFILL(<i>commarea,editdef,databuffer</i>); Example: ZEROFILL(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL ZEROFILL(<i>c(1),e\$,b\$</i>) Example: CALL ZEROFILL(C1(1),E\$,B1\$)

NUMERIC RANGE EDIT (NRANGE)

NRANGE allows you to check the data in the input field to determine if the data is numeric, *and* falls within a range that you have specified in the edit specifications table. The procedure scans the input field and calls the ZEROFILL edit procedure to ensure that all the data is numeric. If the data fails the ZEROFILL edit, the status word is set to minus one, and the edit is complete. If the data passes the ZEROFILL edit, it is then compared with the contents of the low range and high range fields in the edit specifications table. The data must be greater than, or equal to the low range, *and* less than, or equal to the high range. If both these conditions are not met, the status word is set to minus one. If both conditions *are* met, the status word is set to zero.

The calling sequence of the NRANGE edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CNRANGE" USING <i>commarea editdef databuffer</i> . Example: CALL "CNRANGE" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL NRANGE(<i>commarea,editdef,databuffer</i>) Example: CALL NRANGE(MYCOMM,MYEDEF,MYDBUFF)
SPL	NRANGE(<i>commarea,editdef,databuffer</i>); Example: NRANGE(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL NRANGE(<i>c(1),e\$,b\$</i>) Example: CALL NRANGE(C1(1),E\$,B1\$)

MODULO 11 CHECK DIGIT CREATE (M11CREATE)

M11CREATE checks the data in the input field to ensure that the data is numeric, and then creates a modulo eleven check digit. The modulo eleven check digit is a value computed by DEL from the numeric value in the input field. It is used in the modulo eleven check digit verify procedure to check the accuracy of the data entered to the terminal. The procedure scans the input field and calls the ZEROFILL edit procedure to ensure that all the data is numeric. If the data fails the ZEROFILL edit, the status word is set to minus one, and the edit is terminated. If the data passes the ZEROFILL edit, an attempt is made to generate a modulo eleven check digit.

If the check digit is generated, it is inserted as the right-most digit of the input field. If a check digit cannot be generated, the status word is set to minus one.

The calling sequence of the M11CREATE edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CM11CREATE" USING <i>commarea editdef databuffer</i> . Example: CALL "CM11CREATE" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL M11CREATE(<i>commarea,editdef,databuffer</i>) Example: CALL M11CREATE(MYCOMM,MYEDEF,MYDBUFF)
SPL	M11CREATE(<i>commarea,editdef,databuffer</i>); Example: M11CREATE(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL M11CREATE(<i>c(1),e\$,b\$</i>) Example: CALL M11CREATE(C1(1),E\$,B1\$)

MODULO 11 CHECK DIGIT VERIFY (M11VERIFY)

M11VERIFY checks the data in the input field to ensure the data is numeric, generate a modulo eleven check digit, and compare the check digit to the right-most digit of the input data. The procedure calls the ZEROFILL edit procedure to ensure that all the data is numeric. If the data fails the ZEROFILL edit, the status word is set to minus one, and the edit is terminated. If the data passes the ZEROFILL edit, a modulo eleven check digit is generated and compared with the right-most digit of the input data. If the check digits are not equal, the status word is set to minus one. If the check digits are equal, the status word is set to zero.

The calling sequence of the M11VERIFY edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CM11VERIFY" USING <i>commarea editdef databuffer</i> . Example: CALL "CM11VERIFY" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL M11VERIFY(<i>commarea,editdef,databuffer</i>) Example: CALL M11VERIFY(MYCOMM,MYEDEF,MYDBUFF)
SPL	M11VERIFY(<i>commarea,editdef,databuffer</i>); Example: M11VERIFY(MYCOMM,MYEDEF,MYDBUFF);
BASIC	CALL M11VERIFY(<i>c(1),e\$,b\$</i>) Example: M11VERIFY(C1(1),E\$,B1\$)

SUMMARY OF DEL EDIT PROCEDURES

Tables 7-1 and 7-2 show the combinations of data editing that can be performed and the DEL procedures that can be used to check for these data characteristics.

Table 7-1. Data Characteristics

A	Alphabetic characters
B	Numeric characters
C	Spaces (anywhere in data, including embedded spaces)
D	Spaces (to right of data only)
E	Spaces (before and/or after data only, not embedded)
F	Numeric range (numeric data only)
G	Modulo 11 check digit create (numeric data only)
H	Modulo 11 check digit verify (numeric data only)

Table 7-2. DEL Edit Procedure Summary

To check for:	Use DEL edit procedure:
A	ALPHAEDIT
A,D	ALPHAFILL
A,B,C	ANEDIT
B	NUMRCEDIT
B,E	ZEROFILL
F	NRANGE
G	M11CREATE
H	M11VERIFY

USING HIGH-LEVEL INTERFACE PROCEDURES

SECTION

VIII

To simplify coding in many applications, DEL provides higher-level procedures that combine some of the form-access procedures (discussed in Section V), terminal-access procedures (discussed in Section VI), and editing procedures (discussed in Section VII). These procedures allow your program to perform the following tasks, each with a single request:

- Display a form on the terminal screen and read the input entered on the form by the terminal user.
- Edit the input data.

DISPLAYING FORM AND READING INPUT

Your application program can display a form on the terminal screen and read data entered by the user on that form with a single call to the `SHOWFORM` procedure. This procedure, in turn, implicitly calls:

- `FINDFORM`, to locate the form in the form file.
- `GETFORM`, to move the form into a program buffer.
- `WRITETERM`, to write the form to the terminal, and
- `READTERM`, to read the user's input data into the same program buffer used by `GETFORM/WRITETERM`.

To minimize coding in your application program, you could use `SHOWFORM` in place of the above procedures to move a form into the program buffer, display it, and read the user's data the first time this sequence of operations is required. Then, for subsequent data-entry operations, you could use `READTERM` calls.

If `SHOWFORM` executes successfully, it sets the DEL status word to zero. If a file-access error occurs during `WRITETERM` or `READTERM` execution, `SHOWFORM` sets the status word to the appropriate `FCHECK` error code; if a file-access error occurs during `FINDFORM` or `GETFORM` execution, however, `SHOWFORM` sets the status word to the sum of the `FCHECK` error code plus the value 1000. (In this way, DEL distinguishes between errors on the terminal file and those on the form file.) If the form cannot be located in the form file, `SHOWFORM` sets the status word to minus one.

NOTE

Before calling `SHOWFORM`, you must call `OPENFORM` and `OPENTERM`, to open the form file and terminal files, respectively.

The calling sequence for SHOWFORM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CSHOWFORM" USING <i>commarea formname nextform formdatabuffer fdlength</i> . Example: CALL "CSHOWFORM" USING MYCOMM MYFORM MYFORMN MYFDBUF LENFD.
FORTRAN	CALL SHOWFORM(<i>commarea,formname,nextform,formdatabuffer,fdlength</i>) Example: CALL SHOWFORM(MYCOMM,MYFORM,MYFORMN,MYFDBUF,LENFD)
SPL	SHOWFORM(<i>commarea,formname,nextform,formdatabuffer,fdlength</i>); Example: SHOWFORM(MYCOMM,MYFORM,MYFORMN,MYFDBUF,LENFD);
BASIC	CALL SHOWFORM(<i>c(1),f1\$,f2\$,b\$,l</i>) Example: CALL SHOWFORM(C1(1),F1\$,F2\$,B\$,L)

The parameters for SHOWFORM are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
<i>formname</i> <i>f1\$</i>	The name of the area that contains the identity of the current form. This is a CHARACTER data item exactly 16 bytes long.
<i>nextform</i> <i>f2\$</i>	The name of the area in which SHOWFORM will place the name of the next form if it is accessing a chained sequence of forms. This is a CHARACTER data item exactly 16 bytes long.
<i>formdatabuffer</i> <i>b\$</i>	The name of the program buffer to which the form is copied from the form file, and to which the user's data is returned from the terminal. This must be a CHARACTER data item with a minimum length of 64 bytes; a greater length may be required, depending on the amount of data to be read from the terminal.
<i>fdlength</i> <i>l</i>	The name of a variable that contains the length of the program buffer <i>formdatabuffer</i> . When SHOWFORM executes, it overwrites this buffer length with the length of the data actually read from the terminal. This must be a one-word numeric data item.

EDITING NEXT INPUT FIELD

After your application calls FINDFORM to locate a form (and initialize the DEL communication area with data pertinent to that form), and calls READTERM to read data from the terminal, it can edit the current input field with a single call to the EDITFIELD procedure. (Alternatively, FINDFORM and READTERM may have been called implicitly by SHOWFORM.) EDITFIELD implicitly calls NEXTEDIT (Section VII) to retrieve the edit specifications for the input field, and then calls the appropriate DEL editing procedures to accomplish the editing. This effectively replaces two explicit procedure calls with one.

NOTE

The EDITFIELD procedure cannot be used to call user-supplied editing procedures.

If any required procedure is not a DEL editing procedure (as defined in Section VII), EDITFIELD sets the status word to minus 3. If the user's input data fails any edit check, EDITFIELD sets the status word to minus 1 and writes the edit specifications that apply to the current input field into the data area *editdef*. If the last edit specification for the last input field has already been accessed, EDITFIELD sets the status word to minus 2, leaves *editdef* unchanged, and re-sets the communication area to reflect the first unprotected field on the form; thus, the form is ready for the entry of new data at its beginning. If EDITFIELD cannot access the form file, the status word is set to the appropriate FCHECK error code.

NOTE

Before calling EDITFIELD, you must have opened the form file and terminal file (with OPENFORM and OPENTERM), located the form and initialized the communication area (with FINDFORM or SHOWFORM), and read the data from the terminal (READTERM or SHOWFORM).

The EDITFIELD calling sequence is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CEDITFIELD" USING <i>commarea editdef databuffer</i> . Example: CALL "CEDITFIELD" USING MYCOMM MYEDEF MYDBUF.
FORTRAN	CALL EDITFIELD(<i>commarea,editdef,databuffer</i>) Example: CALL EDITFIELD(MYCOMM,MYEDEF,MYDBUF)
SPL	EDITFIELD(<i>commarea,editdef,databuffer</i>); Example: EDITFIELD(MYCOMM,MYEDEF,MYDBUF);
BASIC	CALL EDITFIELD(<i>c(1),e\$,b\$</i>) Example: CALL EDITFIELD(C1(1),E\$,B\$)

The EDITFIELD parameters are:

PARAMETER	MEANING
<i>commarea</i> <i>c</i>	The name of the DEL communication area used for your program. (See OPENFORM discussion in Section V.)
<i>editdef</i> <i>e\$</i>	The name of the area to which EDITFIELD returns the edit specifications for an input field that fails edit-checking. This must be a CHARACTER data item that is exactly 72 bytes long.
<i>databuffer</i> <i>b\$</i>	The name of the program buffer to receive the user's input from the terminal (via READTERM). This buffer can be any length, depending on the number of input characters expected. It must be a CHARACTER data item.

WRITING PROGRAMS

SECTION

IX

DEL application programs can be written in COBOL, FORTRAN, BASIC, or SPL language to serve a variety of data-entry applications. Examples of two such applications are presented in this section, to illustrate some of the programming techniques used for interfacing with DEL. The first example is written in COBOL and the second is written in BASIC.

COBOL PROGRAM

The COBOL program runs the SALES TRANSACTION application introduced in Section III and uses the form shown as an example in that section. The program locates the form (named SALESFORM) in the form file (named SALESFIL), writes the form to the terminal screen, and reads the data entered by the user at the terminal keyboard. If the user enters "XX" in the first input field, the program terminates. But, if he enters any other characters in that field, the program continues. When the user has entered all data on the form and pressed the ENTER key, the program edits all input fields. If any editing procedure fails, the program produces an audible "beeping" sound at the terminal and causes the first erroneous field to blink. When the user corrects the data in this field, the program reads and edits this data again. If any other field contains an error, the same beeping/blinking takes place again for that field. When the user has corrected all erroneous fields, the program writes the valid record to a data file (named MYDATA-FILE) and clears the input fields on the form. When the user enters the next data, the program performs the same reading/editing as before. The program continues until the user enters "XX" in the first input field on the form.

A flow-chart showing the major steps in the application program appears in Figure 9-1. Certain error-checking steps, such as file or terminal-access error detection, are omitted from the chart so that the primary logic is clearly emphasized. All steps, however, appear in the listing of the compiled program shown in Figure 9-2. A symbol table map for this program appears in Figure 9-3.

The program was entered into the system with the Editor subsystem (EDIT/3000) and compiled with the COBOL/3000 compiler. For further information about using these subsystems, please see:

- *EDIT/3000 Reference Manual*
- *COBOL/3000 Reference Manual*

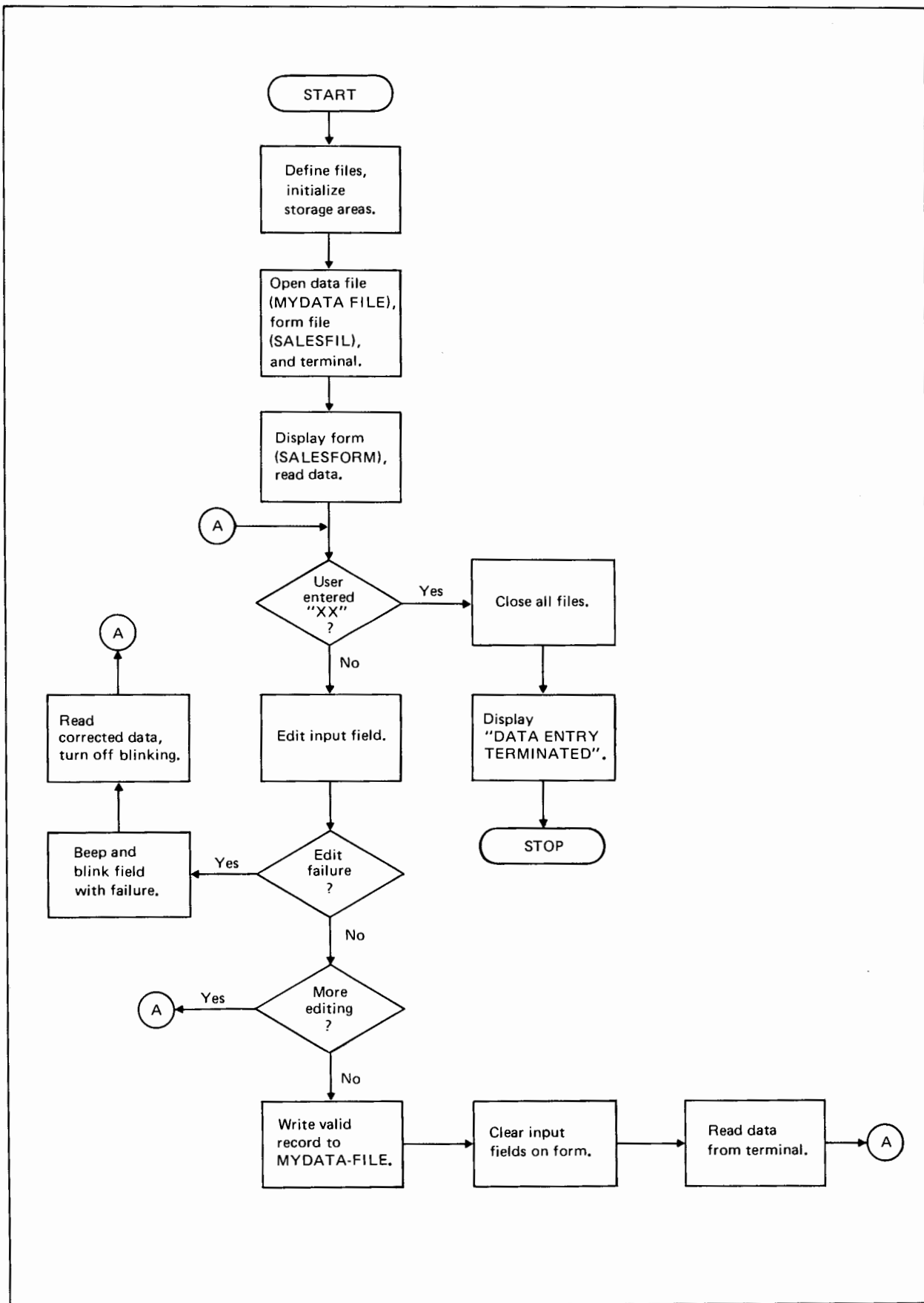


Figure 9-1. COBOL Program Flowchart

```

001000$CONTROL USLINIT,MAP <<INITIALIZES OBJECT FILE, REQUESTS MAP.>>
002000*
003000*
004000* * * * *
005000* IDENTIFICATION DIVISION NAMES PROGRAM AND DATE WRITTEN. *
006000* * * * *
007000*
008000*
009000 IDENTIFICATION DIVISION.
010000 PROGRAM-ID. SALESPRG.
011000 DATE-WRITTEN. MAY 11, 1976.
012000*
013000*

```

```

015000* * * * *
016000* ENVIRONMENT DIVISION NAMES SOURCE AND OBJECT COMPUTERS *
017000* AND INPUT/OUTPUT FILES USED BY PROGRAM. *
018000* * * * *
019000*
020000*
021000 ENVIRONMENT DIVISION.
022000 CONFIGURATION SECTION.
023000 SOURCE-COMPUTER.
024000 HP-3000.
025000 OBJECT-COMPUTER.
026000 HP-3000.
027000 INPUT-OUTPUT SECTION.
028000 FILE-CONTROL.
029000*
030000**** DEFINES DATA FILE TO CONTAIN USER'S INPUT DATA.
031000**** DEL AUTOMATICALLY DEFINES FORM FILE AND TERMINAL FILE.
032000 SELECT MYDATA-FILE ASSIGN TO "DATAFILE".
033000*
034000*

```

Figure 9-2. COBOL Program Listing

```

036000* * * * *
037000*   DATA DIVISION DESCRIBES FILES, RECORDS, BUFFERS, AND OTHER *
038000*   STORAGE AREAS USED BY PROGRAM. *
039000* * * * *
040000*
041000*
042000 DATA DIVISION.
043000 FILE SECTION.
044000*
045000**** DEFINE RECORDS FOR DATA FILE.
046000 FD   MYDATA-FILE
047000       RECORD CONTAINS 423 CHARACTERS
048000       LABEL RECORD IS OMITTED
049000       DATA RECORD IS MYDATA-REC.
050000 01   MYDATA-REC          PIC X(423).
051000 WORKING-STORAGE SECTION.
052000*
053000**** DEFINE FORM FILE NAME (USED IN "OPENTERM").
054000 77   MYFORMFL          PIC X(35) VALUE SPACES.
055000*
056000**** DEFINE FORM NAME (USED IN "CSHOWFORM"/"CFINDFORM").
057000 77   MYFORM          PIC X(16) VALUE SPACES.
058000*
059000**** DEFINE NEXT FORM NAME (USED IN "CSHOWFORM"/"CFINDFORM").
060000**** THIS IS A "DUMMY PARAMETER" SINCE NO OTHER FORM IS USED.
061000 77   NEXTF          PIC X(16) VALUE SPACES.
062000*
063000**** DEFINE TERMINAL FILE (USED IN "COPENFORM").
064000 77   MYTERMFL        PIC X(8) VALUE SPACES.
065000*
066000**** DEFINE BUFFER LENGTHS (USED IN "CSHOWFORM"/"CREADTERM"/
067000**** "CWRITETERM"/"CFINDFORM").
068000 77   LENFD          PIC S9999 USAGE COMP.
069000*
070000 77   LENERRBUF        PIC S9999 USAGE COMP.
071000*
072000 77   LENFORM          PIC S9999 USAGE COMP.
073000*
074000**** DEFINE DATA-ITEM TO CONTAIN FCHECK ERROR CODE FOR OUTPUT.
075000 77   ERROR-CODE        PIC -9999.
076000*
077000**** DEFINE FORM BUFFER.
078000 01   MYFRUF          PIC X(450).
079000*

```

Figure 9-2. COBOL Program Listing (Continued)

```
081000**** DEFINE DATA BUFFER.
082000**** PROGRAM ASSUMES THAT INPUT WILL BE ENTERED
083000**** IN ALL FIELDS ON THE FORM, AND THAT A TOTAL OF FIVE
084000**** TRANSACTIONS PER FORM WILL BE ENTERED.
085000 01 MYDBUF REDEFINES MYFBUF.
086000      05 DB-MO                PIC X(2).
087000      05 DB-DA                PIC X(2).
088000      05 DB-YR                PIC X(2).
089000      05 TRANS-NO             PIC X(6).
090000      05 CUST-ID              PIC X(6).
091000      05 BILLED-NAME          PIC X(25).
092000      05 BILLED-COMP          PIC X(25).
093000      05 SALES-ID             PIC X(6).
094000      05 BILLED-STREET        PIC X(25).
095000      05 BILLED-CITY          PIC X(25).
096000      05 PRODNO-1            PIC X(4).
097000      05 DESCRIP-1           PIC X(45).
098000      05 PRICEDOLS-1          PIC 9999.
099000      05 PRICECNTS-1          PIC 99.
100000     05 QUANTITY-1           PIC X(4).
101000     05 PRODNO-2            PIC X(4).
102000     05 DESCRIP-2           PIC X(45).
103000     05 PRICEDOLS-2          PIC 9999.
104000     05 PRICECNTS-2          PIC 99.
105000     05 QUANTITY-2           PIC X(4).
106000     05 PRODNO-3            PIC X(4).
107000     05 DESCRIP-3           PIC X(45).
108000     05 PRICEDOLS-3          PIC 9999.
109000     05 PRICECNTS-3          PIC 99.
110000     05 QUANTITY-3           PIC X(4).
111000     05 PRODNO-4            PIC X(4).
112000     05 DESCRIP-4           PIC X(45).
113000     05 PRICEDOLS-4          PIC 9999.
114000     05 PRICECNTS-4          PIC 99.
115000     05 QUANTITY-4           PIC X(4).
116000     05 PRODNO-5            PIC X(4).
117000     05 DESCRIP-5           PIC X(45).
118000     05 PRICEDOLS-5          PIC 9999.
119000     05 PRICECNTS-5          PIC 99.
120000     05 QUANTITY-5           PIC X(4).
121000*
122000*
```

Figure 9-2. COBOL Program Listing (Continued)

```
124000**** DEFINE DEL COMMUNICATION AREA.
125000 01 MYCOMM.
126000     05 STATUS-WORD          PIC S9999 USAGE COMP.
127000     05 FILLER              PIC X(254).
128000*
129000**** DEFINE EDITING ERROR-SIGNAL OUTPUT BUFFER.
130000**** WHERE BLANKS APPEAR IN VALUE ITEMS, EITHER
131000**** ESCAPE OR CONTROL CHARACTERS ARE ENTERED.
132000**** THESE ARE NON-PRINTING CHARACTERS. (FIRST
133000**** "FILLER" CONTAINS "ESC&a". "BELLS" CON-
134000**** TAINS THREE CONTROL-G'S.)
135000 01 ERROR-OUTPUT.
136000     05 FILLER              PIC X(3) VALUE IS " &a".
137000     05 ROW                PIC X(3).
138000     05 FILLER              PIC X(1) VALUE IS "r".
139000     05 COL                PIC X(3).
140000     05 FILLER              PIC X(1) VALUE IS "c".
141000     05 BLINKER             PIC X(4).
142000     05 BELLS              PIC X(3) VALUE IS "  ".
143000*
144000**** DEFINE EDITING-ERROR LOCATION DATA.
145000 01 MYEDEF.
146000     05 MYROW              PIC X(3).
147000     05 MYCOL              PIC X(3).
148000     05 FILLER              PIC X(66) VALUE SPACES.
149000*
150000*
```

Figure 9-2. COBOL Program Listing (Continued)

```

152000* * * * *
153000*   PROCEDURE DIVISION SPECIFIES PROGRAM OPERATIONS.   *
154000* * * * *
155000*
156000*
157000 PROCEDURE DIVISION.
158000*
159000**** OPEN ALL FILES USED BY THE PROGRAM.
160000 010-OPEN-FILES.
161000     OPEN OUTPUT MYDATA-FILE.
162000     MOVE "SALESFIL" TO MYFORMFL.
163000     CALL "COPENFORM" USING MYCOMM MYFORMFL.
164000     IF STATUS-WORD = 0 GO TO 025-CHECK-TERM.
165000     IF STATUS-WORD NOT = 0 GO TO 090-FFILE-ERRA.
166000 025-CHECK-TERM.
167000     MOVE "TERMINAL" TO MYTERMFL.
168000     CALL "COPENTERM" USING MYCOMM MYTERMFL.
169000*
170000**** CHECK TERMINAL STATUS.
171000     IF STATUS-WORD = 0 GO TO 030-DISPLAY-AND-READ.
172000     IF STATUS-WORD = - 1 DISPLAY "TERMINAL NOT 2640/2644."
173000         GO TO 075-SHUTDOWN.
174000     IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
175000         DISPLAY "TERMINAL ACCESS ERROR"
176000         ERROR-CODE
177000         GO TO 075-SHUTDOWN.
178000*
179000**** DISPLAY FORM AND READ DATA, FIRST TIME.
180000 030-DISPLAY-AND-READ.
181000*
182000**** SET FORM NAME AND BUFFER LENGTH.
183000     MOVE "SALESFORM" TO MYFORM.
184000     MOVE 450 TO LENFD.
185000     CALL "CSHOWFORM" USING MYCOMM MYFORM NEXTF MYFBUF LENFD.
186000*
187000**** CHECK TERMINAL STATUS AGAIN.
188000 035-STATUS-CHECK.
189000* * * * *
190000* " X H J" IN A DISPLAY STATEMENT IS ESCAPE SEQUENCE THAT *
191000* TURNS OFF FORMAT MODE, MOVES CURSOR TO "HOME" POSITION *
192000* AND CLEARS SCREEN. (THE ESC KEY IS A NON-PRINTING CHAR- *
193000* ACTER.) *
194000* * * * *
195000     IF STATUS-WORD = 0 GO TO 040-EDIT-INPUT.
196000     IF STATUS-WORD = -1
197000         DISPLAY " X H JFILE " MYFORMFL
198000         " DOES NOT CONTAIN " MYFORM
199000         GO TO 075-SHUTDOWN.
200000     IF STATUS-WORD < 1000 GO TO 095-TERM-ERRA.
201000     IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
202000         DISPLAY " X H JCANNOT READ FORM FILE"
203000         MYFORMFL
204000         GO TO 075-SHUTDOWN.
205000*

```



Figure 9-2. COBOL Program Listing (Continued)

```

207000*
208000**** BEGIN EDITING FIELDS.
209000 040-EDIT-INPUT.
210000*
211000**** IF USER ENTERS "XX" IN FIRST FIELD, END PROGRAM.
212000     IF DB-MO = "XX" GO TO 075-SHUTDOWN.
213000*
214000**** PROCEED WITH EDIT.
215000     CALL "CEDITFIELD" USING MYCOMM MYEDEF MYDBUF.
216000     IF STATUS-WORD = 0
217000         GO TO 040-EDIT-INPUT.
218000     IF STATUS-WORD = -2
219000         GO TO 060-VALID-RECORD.
220000     IF STATUS-WORD = -3
221000         GO TO 200-INVALID-PROCNAME.
222000     IF STATUS-WORD > 0
223000         GO TO 100-FF-READFAIL.
224000*
225000**** BEEP AND BLINK FIELDS THAT FAIL EDIT CHECK.
226000     MOVE MYROW TO ROW.
227000     MOVE MYCOL TO COL.
228000     MOVE " &dC" TO BLINKER.
229000     MOVE 18 TO LENERRBUF.
230000     CALL "CWRTITERM" USING MYCOMM ERROR-OUTPUT LENERRBUF.
231000*
232000**** READ CORRECTED INPUT, RE-SET EDIT SPECS, AND
233000**** CONTINUE EDIT CHECKING.
234000     MOVE 450 TO LENFD.
235000     CALL "CREADTERM" USING MYCOMM MYDBUF LENFD.
236000     CALL "CFINDFORM" USING MYCOMM MYFORM LENFORM NEXTF.
237000*
238000**** TURN OFF BLINKING FIELD.
239000     MOVE 15 TO LENERRBUF.
240000     MOVE " &dR" TO BLINKER.
241000     CALL "CWRTITERM" USING MYCOMM ERROR-OUTPUT LENERRBUF.
242000     GO TO 040-EDIT-INPUT.
243000*
244000 060-VALID-RECORD.
245000*
246000**** SEND VALID RECORD TO DATA FILE.
247000     MOVE MYDBUF TO MYDATA-REC.
248000     WRITE MYDATA-REC.
249000*
250000**** DISPLAY ANOTHER FORM, READ NEXT INPUT RECORD.
251000 065-DISPLAY-READ-LOOP.
252000*
253000**** TURN ON FORMAT MODE, HOME CURSOR, AND CLEAR INPUT FIELDS.
254000     MOVE " W H J" TO MYFBUF.
255000     MOVE 6 TO LENFD.
256000     CALL "CWRTITERM" USING MYCOMM MYDBUF LENFD.
257000     IF STATUS-WORD NOT = 0 GO TO 095-TERM-ERRA.
258000     MOVE 450 TO LENFD.
259000*

```

Figure 9-2. COBOL Program Listing (Continued)

```
261000**** READ NEXT INPUT RECORD
262000      CALL "CREADTERM" USING MYCOMM MYDBUF LENFD.
263000      IF STATUS-WORD > 0 GO TO 095-TERM-ERRA.
264000      GO TO 035-STATUS-CHECK.
265000*
266000**** CLOSE ALL FILES USED BY THE PROGRAM.
267000 075-SHUTDOWN.
268000      CALL "CCLOSEFORM" USING MYCOMM.
269000      CALL "CCLOSETERM" USING MYCOMM.
270000      CLOSE MYDATA-FILE.
271000*
272000**** DISPLAY TERMINATION MESSAGES
273000      DISPLAY "DATA ENTRY TERMINATED."
274000*
275000**** TERMINATE PROGRAM.
276000      STOP RUN.
277000*
278000**** ERROR-HANDLING.
279000*
280000**** IF PROGRAM CANNOT OPEN FORM FILE, TRANSMIT APPRO-
281000**** PRIATE MESSAGE.
282000 090-FFILE-ERRA.
283000      IF STATUS-WORD < 0 DISPLAY " X H J" MYFORMFL
284000                                " IS NOT A FORM FILE".
285000      GO TO 075-SHUTDOWN.
286000      IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
287000      DISPLAY " H X JOPEN FAILURE " ERROR-CODE
288000      GO TO 075-SHUTDOWN.
289000*
290000**** IF PROGRAM CANNOT OPEN TERMINAL FILE, TRANSMIT
291000**** MESSAGE.
292000 095-TERM-ERRA.
293000      MOVE STATUS-WORD TO ERROR-CODE.
294000      DISPLAY " X H JTERMINAL ACCESS ERROR" ERROR-CODE
295000      GO TO 075-SHUTDOWN.
296000*
297000**** IF PROGRAM CANNOT READ FORM FILE, TRANSMIT MESSAGE.
298000 100-FF-READFAIL.
299000      MOVE STATUS-WORD TO ERROR-CODE.
300000      DISPLAY " X H JCANNOT READ FORM FILE"
301000      MYFORMFL
302000      GO TO 075-SHUTDOWN.
303000*
304000**** IF EDIT PROCEDURE NAME IS NOT VALID, TRANSMIT
305000**** MESSAGE.
306000 200-INVALID-PROCNAME.
307000      DISPLAY " X H UINVALID DEL PROCEDURE NAME W"
308000      GO TO 075-SHUTDOWN.
```

Figure 9-2. COBOL Program Listing (Continued)

LVL	SOURCE NAME	BASE	DISPL	SIZE	USAGE	CATEGORY	R	O	D	J	BZ
-----	-------------	------	-------	------	-------	----------	---	---	---	---	----

FILE SECTION

FD	MYDATA-FILE				SEQ						
01	MYDATA-REC	DB	000106	000647	DISP	AN					

WORKING-STORAGE SECTION

77	MYFORMFL	DB	000756	000043	DISP	AN					
77	MYFORM	DB	001022	000020	DISP	AN					
77	NEXTF	DB	001042	000020	DISP	AN					
77	MYTERMFL	DB	001062	000010	DISP	AN					
77	LENFD	DB	001072	000002	COMP	N					
77	LENERRBUF	DB	001074	000002	COMP	N					
77	LENFORM	DB	001076	000002	COMP	N					
77	ERROR-CODE	DB	001100	000005	DISP	NE					
01	MYFBUF	DB	001106	000702	DISP	AN					
01	MYDRUF	DB	001106	000643		GROUP		R			
05	DB-MO	DB	001106	000002	DISP	AN					
05	DB-DA	DB	001110	000002	DISP	AN					
05	DB-YR	DB	001112	000002	DISP	AN					
05	TRANS-NO	DB	001114	000006	DISP	AN					
05	CUST-ID	DB	001122	000006	DISP	AN					
05	BILLED-NAME	DB	001130	000031	DISP	AN					
05	BILLED-COMP	DB	001161	000031	DISP	AN					
05	SALES-ID	DB	001212	000006	DISP	AN					
05	BILLED-STREET	DB	001220	000031	DISP	AN					
05	BILLED-CITY	DB	001251	000031	DISP	AN					
05	PRODNO-1	DB	001302	000004	DISP	AN					
05	DESCRIP-1	DB	001306	000055	DISP	AN					
05	PRICEDOLS-1	DB	001363	000004	DISP	DISP-N					
05	PRICECNTS-1	DB	001367	000002	DISP	DISP-N					
05	QUANTITY-1	DB	001371	000004	DISP	AN					
05	PRODNO-2	DB	001375	000004	DISP	AN					
05	DESCRIP-2	DB	001401	000055	DISP	AN					
05	PRICEDOLS-2	DB	001456	000004	DISP	DISP-N					
05	PRICECNTS-2	DB	001462	000002	DISP	DISP-N					
05	QUANTITY-2	DB	001464	000004	DISP	AN					
05	PRODNO-3	DB	001470	000004	DISP	AN					
05	DESCRIP-3	DB	001474	000055	DISP	AN					
05	PRICEDOLS-3	DB	001551	000004	DISP	DISP-N					
05	PRICECNTS-3	DB	001555	000002	DISP	DISP-N					
05	QUANTITY-3	DB	001557	000004	DISP	AN					
05	PRODNO-4	DB	001563	000004	DISP	AN					
05	DESCRIP-4	DB	001567	000055	DISP	AN					
05	PRICEDOLS-4	DB	001644	000004	DISP	DISP-N					
05	PRICECNTS-4	DB	001650	000002	DISP	DISP-N					
05	QUANTITY-4	DB	001652	000004	DISP	AN					
05	PRODNO-5	DB	001656	000004	DISP	AN					
05	DESCRIP-5	DB	001662	000055	DISP	AN					
05	PRICEDOLS-5	DB	001737	000004	DISP	DISP-N					
05	PRICECNTS-5	DB	001743	000002	DISP	DISP-N					

Figure 9-3. COBOL Program Symbol Table Map

LVL	SOURCE NAME	BASE	DISPL	SIZE	USAGE	CATEGORY	R	O	D	J	BZ
05	QUANTITY-5	DB	001745	000004	DISP	AN					
01	MYCOMM	DB	002010	000400		GROUP					
05	STATUS-WORD	DB	002010	000002	COMP	N					
05	FILLER	DB	002012	000376	DISP	AN					
01	ERROR-OUTPUT	DB	002410	000022		GROUP					
05	FILLER	DB	002410	000003	DISP	AN					
05	ROW	DB	002413	000003	DISP	AN					
05	FILLER	DB	002416	000001	DISP	AN					
05	COL	DB	002417	000003	DISP	AN					
05	FILLER	DB	002422	000001	DISP	AN					
05	BLINKER	DB	002423	000004	DISP	AN					
05	BELLS	DB	002427	000003	DISP	AN					
01	MYEDEF	DB	002432	000110		GROUP					
05	MYROW	DB	002432	000003	DISP	AN					
05	MYCOL	DB	002435	000003	DISP	AN					
05	FILLER	DB	002440	000102	DISP	AN					

SOURCE NAME	S/P	INTERNAL NAME	PB-RELATIVE LOC	PRIORITY NO.
010-OPEN-FILES	P	010OPENFILES00'	000000	0
025-CHECK-TERM	P		000103	0
030-DISPLAY-AND-READ	P		000336	0
035-STATUS-CHECK	P		000413	0
040-EDIT-INPUT	P		000670	0
060-VALID-RECORD	P		001215	0
065-DISPLAY-READ-LOOP	P		001244	0
075-SHUTDOWN	P		001375	0
090-FFILE-ERRA	P		001445	0
095-TERM-ERRA	P		001647	0
100-FF-READFAIL	P		001741	0
200-INVALID-PROCNAME	P		002033	0

DATA AREA IS 8002156 WORDS.
 CPU TIME = 0:00:13. WALL TIME = 0:02:54.
 END COBOL/3000 COMPILATION. NO ERRORS. NO WARNINGS.

Figure 9-3. COBOL Program Symbol Table Map (continued)

BASIC PROGRAM

The BASIC program prompts a user at a terminal for the names of a form file, a data file, and a form. The program then reads this input and opens the form, data, and terminal files and locates the form in the form file. It next displays the form and reads the data that the user enters upon it. If the first three characters of this data are "END", the program closes all files and terminates. If they are any other characters, however, the program edits the input. (All edit procedures called must be DEL procedures.) If any edit procedure fails, the program causes the erroneous field to blink and prints an error message at the bottom of the screen. When all edited fields have been successfully entered, the program writes the input data to the data file, chains to the next form in the series (if any), and reads/edits again. This continues until the user enters "END" as the first three characters on a form.

A flow-chart of this program appears in Figure 9-4, and a program listing is shown in Figure 9-5. The program was entered and run via the BASIC Interpreter, described in detail in:

- *BASIC Interpreter Reference Manual*

PREPARING PROGRAMS

When you prepare a DEL application program prior to execution, it is advisable to specify a maximum stack (Z-DL) area size of at least 10,000 words. (If you fail to do this at the preparation stage, the user must do it at the execution stage.) As an example, to prepare the compiled program text in the user subprogram library (USL) file SALESUSL into the program file SALESPRG, specifying a stack area size of 10,000 words, enter the following MPE :PREP command:

```
:PREP SALESUSL,SALESPRG; MAXDATA=10000
```

Optionally, you can save the program file permanently in the system by entering:

```
:SAVE SALESPRG
```

EXECUTING PROGRAMS

When executing a DEL application program, the user enters the MPE :RUN command as follows:

```
:RUN SALESPRG
```

If you did not specify a maximum stack area size of 10000 words when you prepared the program, the user must do so at execution time as follows:

```
:RUN SALESPRG; MAXDATA=10000
```

The user must run the program in BLOCK MODE. If he does not, he will be prompted by the following message when the program begins execution:

```
DEPRESS BLOCK MODE KEY
```

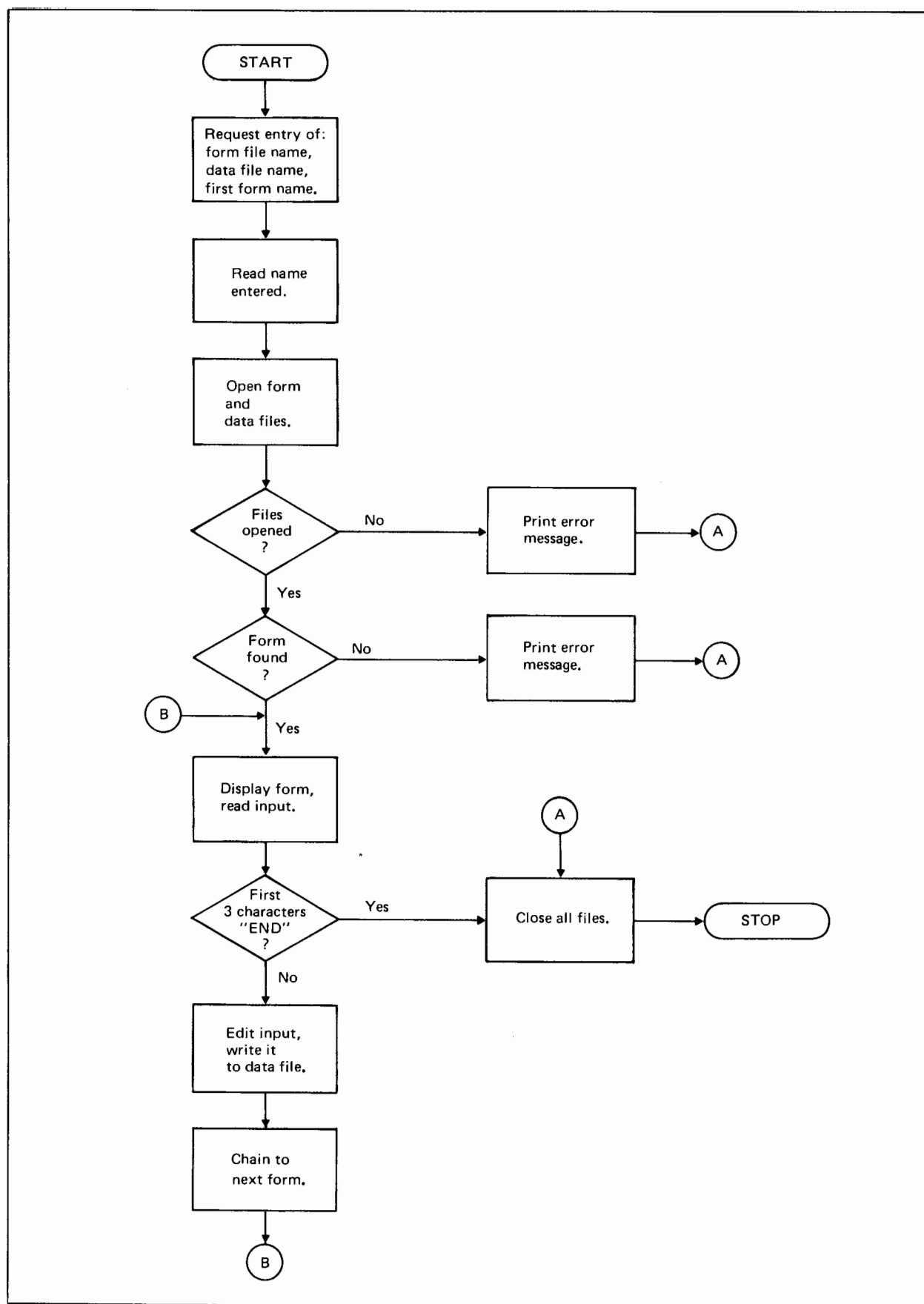


Figure 9-4. BASIC Program Flowchart

```

BASICDE
10 INTEGER C1[128],L1,L2
20 DIM F$(35),F1$(16),F2$(16),B$(255),E$(72),T$(35)
30 DIM W$(255),S$(28)
40 FILES *
50 REM
60 REM INITIALIZE FILE AND FORM NAMES
70 REM
80 F$="DEFORMS "
90 T$="A2640 "
100 F1$="DATAENTRYSTARTUP"
110 F2$="DATAENTRYSTARTUP"
120 REM
130 REM INITIALIZE ARRAYS AND STRINGS
140 REM
150 FOR I=1 TO 128
160   C1[I]=0
170 NEXT I
180 W$[1,255]="*"
190 B$[1,255]="*"
200 E$[1,72]="*"
210 S$[1,28]="*"
220 REM
230 REM OPEN THE TERMINAL FILE
240 REM
250 CALL OPENTERM(C1[1],T$)
260 IF C1[1]=0 THEN 330
270 IF C1[1]<0 THEN 310
280 PRINT "27\"X"27\"H"27\"JCANNOT OPEN TERMINAL FILE ";T$
290 PRINT "ERROR CODE = ";C1[1]
300 STOP
310 PRINT "27\"X"27\"H"27\"J";T$;" IS NOT A 2640"
320 STOP
330 REM
340 REM OPEN FORMS FILE
350 REM
360 CALL OPENFORM(C1[1],F$)
370 IF C1[1]=0 THEN 470
380 IF C1[1]<0 THEN 420
390 PRINT "27\"X"27\"H"27\"JCANNOT OPEN FORM FILE ";F$
400 PRINT "ERROR CODE = ";C1[1]
410 GOTO 2130
420 PRINT "27\"X"27\"H"27\"J";F$;" IS NOT A FORMS FILE"
430 GOTO 2130
440 REM
450 REM LOCATE FORM DEFINITION
460 REM
470 CALL FINDFORM(C1[1],F1$,L1,F2$)
480 IF C1[1]=0 THEN 550
490 IF C1[1]<0 THEN 530
500 PRINT "27\"X"27\"H"27\"JCANNOT READ FORM FILE ";F$
510 PRINT "ERROR CODE = ";C1[1]
520 GOTO 2120
530 PRINT "27\"X"27\"H"27\"JFORM FILE ";F$;" DOES NOT CONTAIN FORM ";F1$
540 GOTO 2120
550 REM
560 REM SET BUFFER SIZE AND GET A BLOCK OF FORM
570 REM
580 L2=255
590 CALL GETFORM(C1[1],B$,L2)

```

Figure 9-5. BASIC Program Listing

```

600 IF C1[1]<>0 THEN 500
610 REM
620 REM UPDATE LENGTH OF FORM AND SEND BLOCK
630 REM
640 L1=L1-L2
650 CALL WRITETERM(C1[1],Bs,L2)
660 IF C1[1]=0 THEN 720
670 PRINT '27"X"'27"H"'27"J"TERMINAL ACCESS ERROR, CODE = ";C1[1]
680 GOTO 2120
690 REM
700 REM IF FORM NOT COMPLETE GET NEXT BLOCK
710 REM
720 IF L1>0 THEN 580
730 REM
740 REM SET BUFFER LENGTH AND READ FROM 2640
750 REM
760 L1=255
770 CALL READTERM(C1[1],Bs,L1)
780 IF C1[1]<>0 THEN 670
790 REM
800 REM IF FORM FILE IS DIFFERENT CLOSE CURRENT
810 REM
820 IF Bs[1;35]=Fs THEN 940
830 CALL CLOSEFORM(C1[1])
840 IF C1[1]=0 THEN 880
850 PRINT '27"X"'27"H"'27"J"CANNOT CLOSE FORM FILE ";Fs
860 PRINT "ERROR CODE = ";C1[1]
870 GOTO 2120
880 Fs=Bs[1;35]
890 CALL OPENFORM(C1[1],Fs)
900 IF C1[1]<>0 THEN 380
910 REM
920 REM OPEN OUTPUT DATA FILE
930 REM
940 Ts=Bs[36;35]
950 ASSIGN Ts,1,X
960 IF X=0 THEN 1020
970 PRINT '27"X"'27"H"'27"J";Ts;" CANNOT BE OPENED, ERROR = ";X
980 GOTO 2130
990 REM
1000 REM SET FORM NAME
1010 REM
1020 F1s=Bs[71;16]
1030 ON END #1 THEN 2080
1040 REM
1050 REM BEGIN DATA ENTRY
1060 REM DISPLAY A FORM AND READ DATA
1070 REM
1080 L1=255
1090 CALL SHOWFORM(C1[1],F1s,F2s,Bs,L1)
1100 IF C1[1]=0 THEN 1200
1110 IF C1[1]>1000 THEN 1150
1120 IF C1[1]>0 THEN 670
1130 PRINT '27"X"'27"H"'27"L";F1s;" IS NOT IN FORM FILE ";Fs
1140 GOTO 2120
1150 C1[1]=C1[1]-1000
1160 GOTO 500
1170 REM
1180 REM CHECK END OF DATA ENTRY
1190 REM

```

Figure 9-5. BASIC Program Listing (Continued)

```

1200 IF B$(1,3)="END" THEN 2120
1210 REM
1220 REM EDIT INPUT DATA
1230 REM
1240 CALL EDITFIELD(C1(1),Es,Bs)
1250 IF C1(1)=0 THEN 1240
1260 IF C1(1)>0 THEN 500
1270 IF C1(1)=-2 THEN 1900
1280 IF C1(1)<>-3 THEN 1350
1290 PRINT "27X"27H"27LEdit ";Es(17,32);" IS NOT PART OF DEL/3000"
1300 GOTO 1240
1310 REM
1320 REM FIELD FAILED EDIT
1330 REM
1340 REM SET-UP LOCK KEYBOARD, CURSOR ADDR & BLINK
1350 Ws(1,5)="27c"27a"
1360 Ws(6,8)=Es(1,3)
1370 Ws(9,9)="r"
1380 Ws(10,12)=Es(4,6)
1390 Ws(13,21)="C"27dC"27a+"
1400 Ws(22,25)=Es(11,14)
1410 Ws(26,32)="C"27X"27d0"
1420 REM TURN OFF FORMAT MODE & MOVE CURSOR TO ERROR MESSAGE
1430 Ws(33,52)="27a23rOCFAILED EDIT "
1440 Ws(53,68)=Es(17,32)
1450 REM TURN ON FORM AT MODE & UNLOCK KEYBOARD
1460 Ws(69,72)="27W"27b"
1470 L1=72
1480 CALL WRITETERM(C1(1),Ws,L1)
1490 IF C1(1)<>0 THEN 670
1500 L1=255
1510 CALL READTERM(C1(1),Ws,L1)
1520 IF C1(1)<>0 THEN 670
1530 REM MOVE FIELD IN ERROR TO INPUT BUFFER & BLANK ERROR MSG
1540 CONVERT Es(7,10) TO X
1550 CONVERT Es(11,14) TO Y
1560 Bs(X;Y)=Ws(X;Y)
1570 Ws(1,14)="27X"27a23rOC"27J"27W"
1580 L2=14
1590 CALL WRITETERM(C1(1),Ws,L2)
1600 IF C1(1)<>0 THEN 670
1610 REM EXECUTE FAILED EDIT AGAIN
1620 IF Es(17,25)<>"ALPHAEDIT" THEN 1660
1630 CALL ALPHAEDIT(C1(1),Es,Bs)
1640 IF C1(1)=0 THEN 1240
1650 GOTO 1350
1660 IF Es(17,25)<>"ALPHAFILL" THEN 1690
1670 CALL ALPHAFILL(C1(1),Es,Bs)
1680 GOTO 1640
1690 IF Es(17,22)<>"ANEDIT" THEN 1720
1700 CALL ANEDIT(C1(1),Es,Bs)
1710 GOTO 1640
1720 IF Es(17,25)<>"NUMRCEDIT" THEN 1750
1730 CALL NUMRCEDIT(C1(1),Es,Bs)
1740 GOTO 1640
1750 IF Es(17,24)<>"ZEROFILL" THEN 1780
1760 CALL ZEROFILL(C1(1),Es,Bs)
1770 GOTO 1640
1780 IF Es(17,22)<>"NRANGE" THEN 1810
1790 CALL NRANGE(C1(1),Es,Bs)

```

Figure 9-5. BASIC Program Listing (Continued)

```

1800 GOTO 1640
1810 IF ES[17,25]<>"M11CREATE" THEN 1840
1820 CALL M11CREATE(C1[1],ES,BS)
1830 GOTO 1640
1840 IF ES[17,25]<>"M11VERIFY" THEN 1240
1850 CALL M11VERIFY(C1[1],ES,BS)
1860 GOTO 1640
1870 REM
1880 REM GOOD RECORD -- WRITE IT TO OUTPUT FILE
1890 REM
1900 PRINT #1;BS;END
1910 REM
1920 REM IF SAME FORM IS TO BE USED AGAIN HOME CURSOR
1930 REM AND USE IT AS IS
1940 IF F1$=F2$ THEN 1970
1950 F1$=F2$
1960 GOTO 1080
1970 BS[1;4]='27"W""27"H"
1980 L1=4
1990 CALL WRITETERM(C1[1],BS,L1)
2000 IF C1[1]<>0 THEN 670
2010 L1=255
2020 CALL READTERM(C1[1],BS,L1)
2030 IF C1[1]<>0 THEN 670
2040 GOTO 1200
2050 REM
2060 REM END OF OUTPUT FILE -- QUIT
2070 REM
2080 PRINT '27"X""27"H""27"JFILE ";TS;" IS FULL"
2090 REM
2100 REM END OF DATA ENTRY CLEAR SCREEN AND CLOSE FILES
2110 REM
2120 CALL CLOSEFORM(C1[1])
2130 CALL CLOSETERM(C1[1])
2140 END

```



Figure 9-5. BASIC Program Listing (Continued)

INPUT/OUTPUT FILES

You may use an MPE :FILE command to equate the formal file designator (name used in your program) for the form file to any actual file designator you desire. However, no :FILE command is necessary for the terminal file, since DEL handles the assignment of this file automatically.

WRITING USER PROCEDURES

When writing your own procedures for use with DEL, remember that any editing information established for the form file is available for these procedures to use in any way you desire. In addition, words 60 through 128 of the DEL communication area, though not used by DEL, are available for user procedures. (See Appendix A.)

LOOK-UP TABLES

In the Editing Specification Display produced by FORMAIN (described in Section III), you are prompted for the name of any file that contains one or more tables to be searched by your application program. This linkage is provided for your convenience. Implementation of the table-search, however, is the responsibility of your program. The linkage is valuable in cases where many edits require split ranges of values, such as:

1000-2000
2500-2699
5000-5133

If this data was stored in a file named PRODMAST, you would enter that file name in response to the prompt as indicated by the underlined entry below:

For file look-up procedures the file name is PRODMAST ← *File name*

The file name is then stored in the edit specifications record (Bytes 41 through 72), and is accessible to the calling program.

In summary, the table look-up manipulation is up to your application program and should be handled in the best manner suitable for that program.

DETECTING/CORRECTING ERRORS

Detection and correction of errors is also the responsibility of your application program. You must consider many variables before selecting the best method for doing this. Two possible methods are presented below; both require enhancing display of the field in error. The WRITETERM procedure should be used to send escape codes and messages.

The first method involves reserving, on the screen, a line for any error messages sent by the application program. A specific line within each screen is preferred but sometimes difficult to maintain between applications. The second method involves inserting a line below the field in error. The escape codes used in both methods appear below. Note that any technique you use to add a line of data to the screen, however temporarily that line may be displayed, risks loss of some information from terminal memory if the memory is full.

To enhance the display of a field that contains an error, enter the line of code shown below:

	column		display		field length	
	row		enhancement			
<code>ESC&a</code>	<code>--r</code>	<code>--C</code>	<code>ESC&d</code>	<code>ESC&a+</code>	<code>--C</code>	<code>ESCX</code>
						<code>ESC&d@</code>
						<code>ESCW</code>
begin cursor addressing	end of row address	end cursor addressing	enhance display	begin cursor addressing	relative positioning	end cursor addressing
					turn off format mode	turn on format mode
					turn off display enhancement	

In the above example, *row*, *column*, and *field length* are available in the DEL Edit Specifications Table. (See Appendix A.)

To display a message on Line 23, enter the function requests shown below:

Function	Entry
Lock the Keyboard	<i>Esc</i> c
Turn off Format Mode	<i>Esc</i> X
Cursor Addressing	<i>Esc</i> &a23rOC
Error Message	“ ”
Turn on Format Mode	<i>Esc</i> W
Unlock Keyboard	<i>Esc</i> b

When you turn on format mode, this automatically returns the cursor to the home position. To move the cursor to the appropriate field automatically, you must specify the necessary row/column positioning. (The terminal user could also use the TAB key to position the cursor.)

To insert an error message on the line below the filled-in error, enter the function requests shown below:

Function	Entry
Lock Keyboard	<i>Esc</i>
Turn off Format Mode	<i>EscX</i>
Cursor Down	<i>EscB</i>
Insert Line	<i>EscL</i> (positioned at beginning of line.)
Error Message	“ ”
Turn on Format Mode	<i>EscW</i>
Unlock Keyboard	<i>Escb</i>

The application program would use the READTERM procedure to return the data back into the buffer. READTERM reads the entire screen. The program could maintain a separate buffer and pick up only the field in error, move it to the original buffer, and continue with the normal program edit cycle.

If the terminal user might also see additional errors (logically-related fields) and correct them, the application program should re-edit the entire buffer contents.

NOTE

A display enhancement that indicates a field in error should be turned off after the editing is complete, a related message should be blanked out, and any inserted line should be deleted. The escape codes would be very similar to those indicated above.

CORRECTING ERRORS

SECTION

X

All error messages appearing on your screen from DEL are generated by the program FORMAIN.T. If an error is caused by incorrect data that you entered, the field containing the data blinks (if your terminal has the blinking feature). FORMAIN.T waits for you to correct and re-enter the data.

Table 10-1 lists the DEL error messages, their meanings and probable causes, and the corrective action you should take. There is also a column indicating which DEL display presents the error message.

There are two messages that may appear before you select any FORMAIN.T function:

1. INPUT DEVICE IS NOT A 2640

FORMAIN.T checks the input device, which must be either an HP 2640 or HP 2644 terminal. If you are not using one of these terminals, the above error message appears and the program terminates.

2. CANNOT OPEN INPUT DEVICE
SYSTEM ERROR CODE = *nnnn*

If FORMAIN.T cannot open your terminal as an input file, the above message appears on the screen and FORMAIN.T terminates. The *nnnn* parameter indicates an error code returned to FORMAIN.T by the MPE intrinsic FCHECK. Refer to *MPE Intrinsic Reference Manual*, (part no. 30000-90010) for meanings of error codes listed under the FCHECK intrinsic.

The message listed below may appear at any time you are using the terminal.

UNRECOVERED TERMINAL ERROR, CODE=*nnnn*.ABORT?

An unrecoverable terminal error occurred. The *nnnn* parameter indicates an error code returned by the MPE intrinsic FCHECK. Refer to *MPE Intrinsic Reference Manual* for meanings of error codes listed under the FCHECK intrinsic. If you enter a YES in response to the prompt ABORT?, FORMAIN.T aborts the current operation you are performing and returns to the Function Selection Display (figure 3-4). You must then restart the operation that was interrupted by the above error message. If you enter any response other than YES to the ABORT? prompt, FORMAIN.T attempts to continue the current operation, ignoring the terminal error.

DATA AREA FORMATS

APPENDIX

A



DEL/3000 COMMUNICATIONS AREA

The DEL communications area contains information that enables DEL to keep track of what form your applications program is currently using, what field of the form is presently being accessed, and what editing applies to this field.

The communications area contains 128 words of contiguous storage as shown in table A-1. In this table, the contents of the communication area are described, and the data type of each word is noted.

Although in most programming applications you will be concerned only with word 1, the DEL/3000 status word, there may be special applications where other words in the communications area are of importance to your program. If, for instance, you want to read data from a terminal, but have not displayed a form from the form file on the terminal, you should:

1. Set word 27 to indicate the number of input (unprotected) fields on the terminal screen.
2. Set word 28 to indicate the *total bytes* to be transferred from the screen. Since there are field separators for each field, calculate the total bytes by adding the total number of fields to the total length of input data characters.

EDITING SPECIFICATIONS TABLES

The DEL Editing Specifications Table (table A-2) denotes the editing to be performed on each unprotected field of the current form. DEL creates a separate table, in the format shown, for each instance where an editing procedure is called by an applications program. If no editing procedures apply to a field, one Editing Specifications Table is created for that field with bytes 15 through 16 set to zero and bytes 17 through 72 remaining blank.

Table A-1. DEL/3000 Communications Area Format

WORD NUMBER	DATA TYPE	CONTENTS
1	Binary	DEL/3000 status word
2	Binary	MPE file identifier for the terminal file
3	Binary	Terminal mode (line/page)
4-11	Binary	Reserved for block mode terminal procedures
12	Binary	MPE file identifier for the forms file
13-20	Character	Current form name
21-22	Binary	First record number of form definition
23	Binary	Length of form definition
24-25	Binary	First record number of edit specifications
26	Binary	Number of edit specifications
27	Binary	Number of input fields in form
28	Binary	Length of input data
29-36	Character	Next form name
37-40	Binary	Reserved for GETFORM
41-42	Binary	Beginning record number of next block of form to be delivered by GETFORM
43	Binary	Length of form definition not yet delivered by GETFORM
44-45	Binary	Number of record containing current edit specifications
46	Binary	Number of edits specified for this form
47	Binary	Number of input fields still to be processed for this form
48	Binary	Number of edits not yet executed for this field
49	Binary	Number of forms in the file
50-58	Binary	Reserved for the forms file procedures
59	Binary	Edit procedure flags
60-128	Binary	Available for user-written input edit procedures

Table A-2. Edit Specifications Table Format

BYTE NUMBER	CONTENTS
1-3	Field location (row number)
4-6	Field location (column number)
7-10	Field location (offset from beginning of input record)
11-14	Field length
15-16	Number of edits for this field
17-32	Edit procedure name
33-34	Test flag number
35-36	Set flag number
37-38	"Same as" flag number
39-40	"Opposite from" flag number
{ 41-56 }	Range (low value)
{ 57-72 }	Range (high value)
or	
{ 41-72 }	File look-up (file name)
or	
{ 41-72 }	Procedure defined data

FORM FILE FORMAT

APPENDIX

B

The form file is a direct access file containing definitions for each form in the file. The format of the file is described in table B-1.

As shown in the table, the first record (record 0) is set aside as a file ID, or identification record, containing data about the entire file. DEL uses this record to verify that the file is a form file.

The following records (records one through n) are directory records containing data on each form in the file; n is determined by dividing the file capacity by 100.

The remaining records are form definition records and input/edit description records. The file contains at least one form definition record for each form in the file, and an input/edit description record for each unprotected field in the form. The input/edit description records are used to create the Edit Specifications Tables as described in Appendix A. If no editing procedures apply to a field, bytes 9 through 10 are set to zero and bytes 11 through 62 are left blank in the input/edit description record.

Tables B-2 through B-5 show the record format of each record in the form file:

Table B-2 describes the format of Record 0, the file ID record.

Table B-3 describes the format of Records 1 through n , the directory records.

Table B-4 describes the format of the form definition records.

Table B-5 describes the format of the edit specifications records.

Table B-1. Forms File Format

RECORD NO.	CONTENTS
Record 0	File ID Record
Records 1 through n $n = \text{file capacity}/100$	Directory Records
Remaining Records	Form Definition Record * Form Definition Record Input/Edit Description Record ** Input/Edit Description Record . . . } Form 1
	Form Definition Record Input/Edit Description Record . . . } Form 2
	Form Definition Record Input/Edit Description Record Input/Edit Description Record Input/Edit Description Record } Form 3
	. . . Additional Forms
* At least one form definition record for each form in the file . **One input/edit description record for each input field in each form.	

Table B-2. File ID Record Format

BYTE NUMBER	DATA TYPE	CONTENTS
1-28	Character	Fully qualified file designator (filename.groupname.accountname)
29-32	Binary	First available record
33-36	Binary	Number of available records
37-38	Binary	Number of forms in file
39-40	Binary	Number of directory records in file
41-44	Binary	First available directory record
45-47	Binary	Not used
48-64	Character	File ID character string



Table B-3. Directory Record Format

BYTE NUMBER	DATA TYPE	CONTENTS
1-16	Character	Form name
17-20	Binary	Record number of first form definition record
21-22	Binary	Length of form definition
23-26	Binary	Record number of first edit specification record
27-28	Binary	Number of edit specifications
29-30	Binary	Number of input fields in form
31-32	Binary	Length of input data
33-48	Character	Next form name (if this form is chained to another form)
49-52	Binary	Date and time created or modified
53-54	Binary	Number of records occupied by this form definition
55-64	Binary	Not used

Table B-4. Form Definition Record Format

BYTE NUMBER	DATA TYPE	CONTENTS
1-64	Character	64 characters of form (blanks deleted where practical)

Table B-5. Input/Edit Description Record Format

BYTE NUMBER	DATA TYPE	CONTENTS
1-2	Binary	Field location (row number)
3-4	Binary	Field location (column number)
5-6	Binary	Field location (offset from beginning of input record)
7-8	Binary	Field length
9-10	Binary	Number of edits for this field
11-26	Character	Edit procedure name
27	Binary	Test flag number
28	Binary	Set flag number
29	Binary	"Same as" flag number
30	Binary	"Opposite from" flag number
{ 31-46 }	Character	Range (low value)
{ 47-62 }	Character	Range (high value)
or		
{ 31-62 }	Character	File look-up (file name)
or		
{ 31-62 }	Character	Procedure defined data

MPE COMMANDS SPECIFICATIONS

APPENDIX

C

The reference specifications for MPE commands :HELLO, :FILE, and :BYE appear in this section. These three MPE commands are the only ones typically used for normal operation of DEL. For information on other MPE commands, refer to the *MPE Commands Reference Manual*.

For each command, the reference specifications show the following information:

- Syntax
- Parameter definitions (including meaning, constraints, and defaults).
- When legal (in sessions or jobs, during break, or programmatically).
- Whether interruptable (with BREAK key).
- Operation or functional description.
- Examples

In the reference specifications, the indication *Available In Break* means: the command can be entered and executed if the *current* operation is suspended by pressing the BREAK key on the terminal (or calling the CAUSEBREAK intrinsic). The notation *Available Programmatically* means: the command is available through the COMMAND intrinsic. CAUSEBREAK and COMMAND are explained in *MPE Intrinsics Reference Manual*. A summary of the rules for entering commands, plus notes on the syntax conventions used in this manual appear in table C-1.

Table C-1. Reference Notes for Command Definitions

ELEMENTS OF COMMAND FORMAT

- Leading Colon: is the prompt/command identifier character in interactive sessions.
 is the command identifier character in batch jobs.
- Command Name: is shown in CAPITAL LETTERS IN REGULAR (ROMAN) TYPE, contains no blanks, is delimited by a non-alphabetic character (usually a blank).
- Parameters: are shown in CAPITAL LETTERS IN REGULAR TYPE when they are literal information that you always enter exactly as shown;
 are shown in *lower-case italics* when they are variable parameters to be replaced by information that you must supply.
- Positional Parameters: have significance implied by positional order after command name; use adjacent commas (or semicolons where required) to indicate omitted parameter(s) as follows:
 - :COMMANDNAME p1,,p3 (from middle of list)
 - :COMMANDNAME ,p2,p3 (from beginning of list)
 - :COMMANDNAME p1 (from end of list)
- Keyword Parameters: are separated by semicolons and can appear in any order.
- Mixed Parameters: positional parameters are given first; first keyword indicates end of positional list.
- Optional Parameters: [A] means "A" may be included.
 [A]
 [B] means "A" or "B" *may* be included.
 { A }
 { B } means "A" or "B" *must* be included.
 [A]
 [B] means "A" and/or "B" may be included in any order.

USER/SYSTEM DIALOGUE

- User input is underlined where necessary for clarity: NEW NAME? ALPHA1

:HELLO

Begins an interactive session.

SYNTAX

```
:HELLO [sessionname,]username[/userpasw].acctname[/acctpasw]
        [,groupname[/grouppasw]]

        [;TERM = termtype]
        [;TIME = cpusecs]

        [;PRI = { BS
                  CS
                  DS
                  ES } ]

        [;INPRI = inputpriority
        ;HIPRI ]
```

PARAMETERS

sessionname Arbitrary name used in conjunction with *username* and *acctname* parameters to form a fully-qualified session identity. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default: null session name. (Optional parameter.)

NOTE

A *fully-qualified session identity* consists of:

[*sessionname*,]*username.acctname*

and furnishes the minimum information required for log-on. Embedded blanks are forbidden in *username.acctname* combination.

username A user name, established by Account Manager, that allows you to log-on under this account. This name is unique within the account. Contains from 1 to 8 alphanumeric characters, beginning with letter. (Required parameter.)

userpasw Your user password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *username* by slash with no surrounding blanks, as in *username/userpasw*. (Required if assigned.)

acctname Name of your account, as established by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter.

NOTE

Must be preceded by period as a delimiter.

(Required parameter.)

acctpasw Account's password, optionally assigned by System Manager. Contains from 1 to 8 alphanumeric characters, beginning with letter. Separated from *acctname* by slash with no surrounding blanks, as in *acctname/acctpasw*. (Required if assigned.)

:HELLO

(Continued)

<i>groupname</i>	Name of file group to be used for local file domain and central-processor time charges, as established by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default: Your home group if you are assigned one by Account Manager. (<i>Optional</i> if you have a home group; <i>Required</i> if you do not.)																				
<i>grouppasw</i>	Group's password, optionally assigned by Account Manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Separated from <i>groupname</i> by slash with no surrounding blanks, as in <i>groupname/grouppasw</i> . (Not needed when you log-on under home group. Otherwise, required if assigned.)																				
<i>termtype</i>	Type of terminal used for input. (MPE uses this parameter to determine device-dependent characteristics such as delay factors for carriage returns.) Must be number from 0 to 11, as follows: <table><tr><td>0</td><td>ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).</td></tr><tr><td>1</td><td>ASR 37 EIA-compatible (10 cps).</td></tr><tr><td>2</td><td>ASR 35 EIA-compatible (10 cps).</td></tr><tr><td>3</td><td>Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).</td></tr><tr><td>4</td><td>HP 2600A or DATAPOINT 3300 (10-240 cps).</td></tr><tr><td>5</td><td>Memorex 1240 (10, 15, 30, 60 cps).</td></tr><tr><td>6</td><td>GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).</td></tr><tr><td>9</td><td>HP 2615A (MiniBee) (10-240 cps).</td></tr><tr><td>10</td><td>HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).</td></tr><tr><td>11</td><td>HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).</td></tr></table>	0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).	1	ASR 37 EIA-compatible (10 cps).	2	ASR 35 EIA-compatible (10 cps).	3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).	4	HP 2600A or DATAPOINT 3300 (10-240 cps).	5	Memorex 1240 (10, 15, 30, 60 cps).	6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).	9	HP 2615A (MiniBee) (10-240 cps).	10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).	11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).
0	ASR 33 EIA-compatible HP 2749B (10 characters per second (cps)).																				
1	ASR 37 EIA-compatible (10 cps).																				
2	ASR 35 EIA-compatible (10 cps).																				
3	Execuport 300 Data Communication Transceiver Terminal (10, 15, 30 cps).																				
4	HP 2600A or DATAPOINT 3300 (10-240 cps).																				
5	Memorex 1240 (10, 15, 30, 60 cps).																				
6	GE Terminet 300 or 1200, or Data Communication Terminal, Model B (10, 15, 30, 120 cps).																				
9	HP 2615A (MiniBee) (10-240 cps).																				
10	HP 2640A or HP 2644 Interactive Display Terminal (when used predominantly in character mode.) (10-240 cps).																				
11	HP 2640A or HP 2644 Interactive Display Terminal. (Recommended if used at speeds exceeding 30 cps when you expect transfers in block/line mode.) (10-240 cps).																				

Default: For hardwired terminals, determined by System Supervisor during system configuration; for terminals that are *not* hardwired, MPE attempts to handle the input/output in the best manner possible based on available information. (Required parameter to ensure correct listings if your terminal is *not* the default *termtype*.)

cpusecs

Maximum central-processor time that your session can use, entered in seconds. When this limit is reached, session is aborted. Must be value from 1 to 32767. To specify no limit, enter question mark or omit this parameter.

Default: no limit. (Optional parameter.)

BS }
CS }
DS }
ES }

The execution priority class that the Command Interpreter uses for your session, and also the default priority for all programs executed within the session. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest permitted for your account or user name by the system, MPE assigns the highest priority possible below BS. Default: CS.

NOTE

DS and ES are intended primarily for batch jobs; their use for sessions is generally discouraged.

(Optional parameter.)

inputpriority

Relative input priority used in checking against access restrictions imposed by the *job fence*, if one exists. Takes effect at log-on time. Must be a value from 1 (lowest priority) to 13 (highest priority). If you supply a value less than or equal to current job fence set by Console Operator, session is denied access.

Default: 8 if logging of session/job initiation is enabled, 13 otherwise. (Optional parameter.)

HIPRI

Request for maximum session-selection input priority, causing session to be scheduled regardless of current job fence or execution limit for sessions.

NOTE

You can specify this parameter only if you have System Manager or Supervisor Capability.

(Optional parameter.)

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Log-on message aborted.)

:HELLO

(Continued)

OPERATION

Initiates an interactive session by establishing contact with MPE. From this point on, you are automatically prompted by the appearance of a colon whenever MPE is ready to accept a command during your session.

NOTE

Your must enter the :HELLO command via a terminal — no other device can be used for this purpose.

If you omit any passwords required in the :HELLO command, MPE prompts you for each of them individually.

When MPE initiates the session, it transmits the following information to your terminal:

SESSION NUMBER = #Snnnn

date, time

HP 32002v.uu.ff

nnnnn Session number assigned by MPE to uniquely identify the session. This may range from one to four digits long.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

v MPE version level. (A letter.)

uu MPE update level. (A number.)

ff MPE fix level. (A number.)

EXAMPLE

To start a session named ALPHA, under the user name MAC, account name TECHPUBS, and file group XGROUP (requiring Password XPASS), enter the following :HELLO command. (Log-on information printed by MPE follows this command.)

```
:HELLO ALPHA,MAC,TECHPUBS,XGROUP/XPASS
SESSION NUMBER = #S241
TUE, AUG 26, 1975, 1:59 PM
HP32002A.00.00
```

Re-defines file characteristics
supplied by FOPEN intrinsic.

SYNTAX

For New or Old (Existing) Files:

```

:FILE formaldesignator
    [ = $NEWPASS
      [ = filereference ] [ , NEW ]
      = $OLDPASS
      [ = filereference ] [ , OLD
                          , OLDTEMP ] ]

[ ;REC = [ resize ] [ , [ blockfactor ] [ , [ F
                                U
                                V ] [ , BINARY
                                , ASCII ] ] ] ] †

[ ;CCTL
  ;NOCCTL ] †

[ ;ACC = { IN
            OUT
            UPDATE
            OUTKEEP
            APPEND
            INOUT
          } ]

[ ;NOBUF
  ;BUF [ = numbuffers ] ]

[ ;EXC
  ;EAR
  ;SHR ]

[ ;MULTI
  ;NOMULTI ] (Series II only)

[ ;MR
  ;NOMR ]

[ ;DEL
  ;SAVE
  ;TEMP ]

[ ;DEV = [ device ] [ , [ outputpriority ] [ , numcopies ] ] ] †
[ ;CODE = filecode ] †
[ ;DISC = [ numrec ] [ , [ numextents ] [ , initalloc ] ] ] †

[ ;NOWAIT
  ;WAIT ] (Series II only)

```



NOTE

The parameter group [;DISC = [*numrec*] [, [*numextents*] [, *initalloc*]] cannot be included if the parameter group

[= *filereference*] [, OLD
OLDTEMP] is specified.

Dagger (†) indicates these parameters are not used for old disc files.

:FILE

(Continued)

SYNTAX (Cont)

For System-Defined Files:

:FILE *formaldesignator* = \$NULL

or

:FILE *formaldesignator* = $\left\{ \begin{array}{l} \$STDIN \\ \$STDINX \\ \$STDLIST \end{array} \right\}$

$\left[\begin{array}{l} ;REC = \left[\begin{array}{l} \text{resize} \end{array} \right] \left[\begin{array}{l} , \left[\begin{array}{l} \text{blockfactor} \end{array} \right] \left[\begin{array}{l} , \left[\begin{array}{l} F \\ U \\ V \end{array} \right] \left[\begin{array}{l} , \text{BINARY} \\ , \text{ASCII} \end{array} \right] \end{array} \right] \end{array} \right]$

$\left[\begin{array}{l} ;CCTL \\ ;NOCCTL \end{array} \right]$

$\left[\begin{array}{l} ;ACC = \left\{ \begin{array}{l} \text{IN} \\ \text{OUT} \\ \text{UPDATE} \\ \text{OUTKEEP} \\ \text{APPEND} \\ \text{INOUT} \end{array} \right\} \end{array} \right]$

$\left[\begin{array}{l} ;NOBUF \\ ;BUF \quad [= \text{numbuffers}] \end{array} \right]$

$\left[\begin{array}{l} ;EXC \\ ;EAR \\ ;SHR \end{array} \right]$

$\left[\begin{array}{l} ;MULTI \\ ;NOMULTI \end{array} \right]$ (Series II only)

$\left[\begin{array}{l} ;MR \\ ;NOMR \end{array} \right]$

$\left[\begin{array}{l} ;NOWAIT \\ ;WAIT \end{array} \right]$ (Series II only)

For User Pre-Defined (Back-Referenced) Files

:FILE *formaldesignator* = **formaldesignator*

NOTE

The second parameter in this command *must* include the preceding asterisk (*) noted above.

:FILE

(Continued)

PARAMETERS

<i>formaldesignator</i>	Formal file designator referenced in your program. This is the name by which your program recognizes the file. Contains from 1 to 8 alphanumeric characters, beginning with a letter. (Required parameter.)
<i>filereference</i>	Actual file designator. This is the name by which MPE recognizes the file, written in the following format: filename[/lockword][.groupname][.accountname] All four sub-parameters are names that contain from 1 to 8 alphanumeric characters, beginning with a letter. If the file has no lockword and belongs to the log-on group and account, only <i>filename</i> is necessary. If you omit the <i>filereference</i> parameter from the :FILE command, the actual designator is equated to the formal designator. (Optional parameter.)
\$NEWPASS	A temporary disc file that can be automatically passed to any succeeding MPE command within the session/job, which references the file by the name \$OLDPASS. When \$NEWPASS is closed, its name is automatically changed to \$OLDPASS, and any previous file named \$OLDPASS in the session/job is deleted. (Optional parameter.)
\$OLDPASS	The name of the last temporary disc file closed as \$NEWPASS. (Optional parameter.)

NOTE

If you do not include *any* actual file designator (*filereference*, \$NEWPASS, or \$OLDPASS) in the :FILE command, the formal designator is used as the actual designator.

NEW	Specification that the file is a new file. (Required parameter if <i>filereference</i> is used for a new file; Optional parameter otherwise.)
OLD	Previously-existing permanent file saved in system file domain. File continues to exist after current session/job terminates. (Optional parameter).
OLDTEMP	Previously-existing temporary file in session/ job temporary file domain. File is deleted at end of current session/job. (Optional parameter.)

NOTE

NEW, OLD, and OLDTEMP specify the file domain indicating where the file exists. If all are omitted, the domain specified in FOPEN intrinsic takes effect.

numbuffers Number of buffers to be allocated for the file. This must be an integer, with a maximum value of 16. If set to 0, a value of 2 is assigned. Do not use this parameter for files representing interactive terminals, since a system-managed buffering method is used in such cases. (Optional parameter.)

NOTE

If *NOBUF* and *numbuffers* are both omitted, a default value of 2 buffers is assigned.

EXC After file is opened, prohibits concurrent access (in any mode) to file through another FOPEN request, whether issued by this or another running program (process), until this process issues FCLOSE or terminates. (Optional parameter.)

EAR After file is opened, prohibits concurrent write-access to file through another FOPEN request within this or another process, until this process issues an FCLOSE or terminates. (Optional parameter.)

SHR After file is opened, permits concurrent access to file through another FOPEN request issued by this or another process in any session or job. (Optional parameter.)

NOTE

If AC=IN is specified, and EXC, EAR, and SHR are omitted, then default assigned is SHR. Otherwise, default assigned is EXC.

MULTI Shares this file so that several concurrently-running programs (processes) *in this session/job* can transfer records to and from the file sequentially, regardless of *blockfactor* or buffering (if any) specified. All access-control information, including buffers themselves, is shared between the processes, with result that transfers occur in the order in which the processes request access. You cannot, however, share the file in this way with processes running under other sessions/jobs. (Optional parameter.) (Series II only)

NOMULTI Prohibits sharing files in MULTI mode. (Optional parameter.) (Series II only)

NOTE

If MULTI and NOMULTI are both omitted, the default assigned is NOMULTI.

MR Request for Multi-record Mode. Individual read or write requests are not confined to record boundaries. Thus, if the number of words or bytes to be transferred exceeds the size of the record, the remaining words or bytes are taken from subsequent records until the number requested are transferred. Restricted to NOBUF files. (Optional parameter.)

NOMR Request for normal recording mode (with no multi-record read/write). Individual read/write requests are confined to record boundaries. (Optional parameter.)

NOTE

If MR and NOMR are both omitted, the default assigned is NOMR.

DEL	Request for regular temporary file disposition; the file is deleted when your program closes it. (Optional parameter.)
SAVE	Request for permanent file disposition; the file remains in the system file domain, potentially available to other users, when your program closes it. (Optional parameter.)
TEMP	Request for session/job temporary file disposition. The file remains in the session/job temporary file domain when your program closes it; but when your session/job terminates, the file is deleted. (Optional parameter.)



NOTE

DEL, SAVE, and TEMP denote the disposition of the file. If none of these keywords is specified in this :FILE command, or the disposition is not specified in the FCLOSE intrinsic that closes the file, the file is returned to the disposition it had when opened. For example, a new file (other than \$NEWPASS) is deleted; an old file is returned to the domain in which it was found.

device

A *device class name* designating the type of device, or a *logical device number* indicating the specific device, on which the file resides.

The device class name makes a non-specific, generic reference to a *type* of device (such as any disc drive or magnetic tape unit). This name is defined and related to a specific set of devices when the system is configured. It must contain from one to eight alphanumeric characters, begin with a letter, and terminate with any non-alphanumeric character such as a blank. Examples are CARD, LP, TTY, and TAPE2.

The logical device number refers to a specific single device. This is a number assigned to each device when the system is generated. This specification is only used when assignment of a particular device is truly necessary. For example, you would specify the logical device number of a specific device when running a hardware diagnostic program for that device.

The device specification is *not* used if the file is an old disc file or if the actual file designator used is \$STDIN, \$STDINX, \$STDLIST, \$NEWPASS, \$OLDPASS, or \$NULL (since these names are already assigned to devices by the system).

If the file is a new disc file and the *device* parameter specifies a device class name, all extents are restricted to devices of this class; if the *device* parameter specifies a logical device number, all extents are restricted to the single device identified by that number. Default: Device Class Name DISC. (Optional parameter.)

:FILE

(Continued)

<i>outputpriority</i>	(Spooled output devicefiles only.) The output priority for this file, used to select next spooled devicefile (on disc) for output, among all those contending for a specific printer or card punch. Determines the order in which files are produced when several are awaiting the same device. This must be a value between 1 (lowest priority) and 13 (highest priority), inclusive. If this value is less than the current <i>outfence</i> set by the Console Operator, the file is deferred from printing/punching until the operator raises the <i>outputpriority</i> of the file or lowers the <i>outfence</i> . This parameter is ignored for non-spooled output devices. Default: 8 (if logging enabled) or 13 (if logging disabled). (Optional parameter.)
<i>numcopies</i>	(Spooled output devicefiles only.) Number of copies of file to be produced by spooling facility. The copies will not appear contiguously if the Console Operator intervenes or if a file of equal or higher <i>outputpriority</i> becomes READY before the last copy is started. This parameter is ignored for non-spooled output devices. Default: 1. (Optional parameter.)
<i>filecode</i>	Code indicating a specially-formatted file. This code is recorded in the file label and is available to programs accessing the file through the FGETINFO intrinsic. Must be a positive integer ranging from 0 to 1023, or one of the HP-defined integers or mnemonics listed below:

Mnemonic	HP-defined Integer	Defines the File as:
USL	1024	User subprogram library (USL) file.
BASD	1025	BASIC data file.
BASP	1026	BASIC program file.
BASFP	1027	BASIC fast program file.
RL	1028	Relocatable library (RL) file.
PROG	1029	Program file.
STAR	1030	STAR file.
SL	1031	Segmented library (SL) file.
(None)	1040	Cross-Loader ASCII file (SAVE).
(None)	1041	Cross-Loader relocated binary file.
(None)	1042	Cross-Loader ASCII file (DISPLAY).
(None)	1050	EDIT KEEPQ file (non-COBOL).
(None)	1051	EDIT KEEPQ file (COBOL).
(None)	1052	EDIT TEXT file (COBOL).
(None)	1060	RJE punch file.
(None)	1070	QUERY procedure file.

Default: 0. (Optional parameter.)

:FILE

(Continued)

numrec Total maximum file capacity, specified only for a NEW file, in terms of logical records (for files containing fixed-length records) and blocks (for files containing variable-length and undefined-length records). Maximum capacity allowed is 2,097,120 sectors. Default: 1023. (Optional parameter.)

numextents Number of extents (integral number of contiguously-located disc sectors) that can be dynamically allocated to the file as logical records are written to it; specified for new files only. The size of each extent (in terms of records) is determined by the *numrecs* parameter value divided by the *numextents* parameter value. Extents can be allocated on any disc in the device class specified in the *device* parameter. If you want to ensure that all extents for a file reside on the same disc, use the logical device number of that disc or a device class name relating to a single disc device, in the *device* parameter.

If specified, *numextents* must be an integer value from 1 to 32. Default: 8. (Optional parameter.)

initalloc Number of extents to be allocated to the file *at the time it is opened*, specified for new files only. Must be an integer from 1 to 32. If attempt to allocate requested space fails, the FOPEN intrinsic that opens the file returns an error condition code to your program. Default: 1. (Optional parameter.)

NOWAIT Allows program that references this file to initiate an input/output request and to have control returned before completion of the request. To confirm completion of such requests, your program must call the IOWAIT intrinsic after each request, as described in *MPE Ininsics Reference Manual*. Also, multi-record access is prohibited.

NOTE

To use the NOWAIT parameter, your program must be running in Privileged Mode. If you specify NOWAIT, this implies that no buffering (NOBUF) and normal recording mode (NOMR) are used. (Optional parameter.) (Series II only)

WAIT Disallows NOWAIT input/output, discussed above. (Optional parameter.) (Series II only)

NOTE

If NOWAIT and WAIT are both omitted, WAIT is assigned by default.

**formaldesignator* Name of a file defined in a previous :FILE command, preceded by an asterisk. (Required parameter.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	YES
	Programmatically?	YES
Breakable?		NO

:FILE

(Continued)

OPERATION

Re-specifies characteristics for a file. Used to override FOPEN intrinsic parameters (or their default values supplied by MPE). Does not, however, override label on old disc file or physical constraints of device on which file resides. Takes effect when subsystem or program opens the file referenced; remains in effect throughout your session or job, unless superseded by a later :FILE command or revoked by a :RESET command. At session/job termination, the :FILE command is cancelled. Applies to files on any devices.

NOTE

A :FILE command deletes *all* specifications appearing in any previous :FILE command that references the same file.

EXAMPLES

A program references two files by the file names (formal file designators) SOURCE and DEST, but you wish to use two disc files recognized in the system by the actual file designators INX and OUTX respectively in place of these files. You can re-specify the file names when you run your program in this fashion:

```
→ :FILE SOURCE = INX
→ :FILE DEST = OUTX
:RUN MYPROG
```

You run a program that reads input from a file it calls INPT and directs output to a file called OUTPT. INPT is a card file (opened as an old file with 80-byte logical records and device class name CARD). OUTPT is a new file on disc, to be assigned an actual file designator FILEX with 64-word, fixed-length logical records written as two records per block in ASCII code. The file may contain a maximum of 800 logical records, is immediately allocated two extents, and may have a maximum of ten extents. When your program closes the file, it is saved as a permanent file in the system file domain. You can specify these files as follows:

```
→ :FILE INPT, OLD; REC = -80; DEV = CARD
→ :FILE OUTPT = FILEX, NEW; REC = 64,2,F,ASCII; DISC=800,10,2; SAVE
:RUN PROGFLX
```

:BYE

Ends an interactive session.

SYNTAX

:BYE

PARAMETERS

None

USE

Available	In Session?	YES
	In Job?	NO
	In Break?	YES
	Programmatically?	NO
Breakable?		YES (Aborts log-off message.)

OPERATION

Terminates a session, resulting in a display of log-off information in the following format:

```
CPU (SEC) = cpusecs
CONNECT (MIN) = connectmin
date, time
END OF SESSION
```

cpusecs Total central-processor time used by the session, in seconds, charged against log-on group and account.

connectmin Total time elapsed between log-on and log-off, rounded upward to nearest minute, charged against log-on group and account.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

EXAMPLE

To terminate the current session, enter :BYE.

```
:BYE

CPU (SEC) = 2
CONNECT (MIN) = 2
TUE, AUG 26, 1975, 2:02 PM
END OF SESSION
```


ASCII CHARACTER SET

APPENDIX

D

The ASCII Character Set/Collating Sequence is shown in table D-1 for your reference.

Table D-1 ASCII Code Chart

BIT 7 6 5 4 3 2 1 0	CONTROL CHARACTERS		DISPLAYABLE CHARACTERS								ESCAPE SEQUENCES							
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0000	NUL	DLE	SP	0	@	P	`	p	SPACE	PRINT	0	α	DELETE CHAR	P				
0001	SOH	DC1	!	1	A	Q	a	q		SET TAB	↑	A	INSERT CHAR ON	Q	a	CURSOR SENSE	f ₁	q
0010	STX	DC2	"	2	B	R	b	r		CLEAR TAB	↓	B	INSERT CHAR OFF	R	b	KEYBOARD ENABLE	f ₂	r
0011	ETX	DC3	#	3	C	S	c	s		#	→	C	ROLL UP	S	c	KEYBOARD DISABLE	f ₃	s
0100	EOT	DC4	\$	4	D	T	d	t		\$	←	D	ROLL DOWN	T	d	ENTER	f ₄	t
0101	ENQ	NAK	%	5	E	U	e	u		%		E	RESET TERMINAL	U	e		f ₅	u
0110	ACK	SYN	&	6	F	V	f	v	PARAMETER SEQUENCE	&		F	NEXT PAGE	V	f		f ₆	v
0111	BEL	ETB	'	7	G	W	g	w				G	PREV PAGE	W	g		f ₇	w
1000	BS	CAN	(	8	H	X	h	x				H	FORMAT MODE ON	X	h		f ₈	x
1001	HT	EM	)	9	I	Y	i	y	DEFINE SET	)		I	FORMAT MODE OFF	Y	i			y
1010	LF	SUB	*	:	J	Z	j	z		*		J	HORIZONTAL TAB	Z	j			z
1011	VT	ESC	+	; K		k	{					K	DISPLAY FUNCTIONS ON		k			{
1100	FF	FS	,	<	L	\	l	:				L	DISPLAY FUNCTIONS OFF	\	l			:
1101	CR	GS	=	=	M	]	m	[				M	ERASE TO END OF LINE	]	m			[
1110	SO	RS	>	>	N	^	n	~				N	DELETE LINE	^	n			~
1111	SI	US	/	? O	-	o	DEL					O	START UNPROTECT FIELD	-	o			DEL

DATA ENTRY LIBRARY FORM LAYOUT SHEET

APPENDIX

E

On the next page is a sample Form Layout Sheet that can be duplicated and used as a worksheet in laying out your DEL forms.

HEWLETT PACKARD

Page _____ of _____

[illegible][illegible]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78

DATA ENTRY LIBRARY EDITING SPECIFICATION SHEET

APPENDIX

F

On the next page is a Data Entry Library Editing Specification Sheet that can be duplicated and used as a worksheet for DEL edit specifications.

DATA ENTRY LIBRARY EDITING SPECIFICATION SHEET

[illegible]

FCHECK ERROR CODES

APPENDIX

G

As indicated in Section X, many DEL error messages include error codes returned by the MPE FCHECK intrinsic. The following list shows the error codes and indicates what kind of error occurred to generate the code.

Code (Decimal)	Meaning
0	End of file.
1	Illegal DB register setting (typically, a request in split-stack mode when it is illegal).
2	Illegal capability
8	Illegal parameter value.
20	Invalid operation.
21	Data parity error.
22	Software time-out.
23	End of tape.
24	Unit not ready.
25	No write ring on tape.
26	Transmission error.
27	Input/output time-out.
28	Timing error or data overrun.
29	Start input/output (SIO) failure.
30	Unit failure.
31	End of line (special character terminator).
32	Software abort of input/output operation.
33	Data lost.
34	Unit not on line.
35	Data set not ready.
36	Invalid disc address.
37	Invalid memory address.
38	Tape parity error.
39	Recovered tape error.
40	Operation inconsistent with access type.
41	Operation inconsistent with record type.
42	Operation inconsistent with device type.
43	The <i>tcount</i> parameter value exceeded the <i>recsize</i> parameter, but the <i>multirecord access aoption</i> was not specified when the file was opened.
44	The FUPDATE intrinsic was called, but the file was positioned at record zero. (FUPDATE must reference the last record read, but no previous record was read.)
45	Privileged file violation.
46	File space on all discs in the device class specified is insufficient to satisfy this request.
47	Input/output error on a file label.
48	Invalid operation due to multiple file access.
49	Unimplemented function.

Code (Decimal)	Meaning
50	The account referenced does not exist.
51	The group referenced does not exist.
52	The referenced file does not exist in the system (permanent) file domain.
53	The referenced file does not exist in the job temporary file domain.
54	The file reference is invalid.
55	The referenced device is not available.
56	The device specification is invalid or undefined.
57	Virtual memory is not sufficient for the file specified.
58	The file was not passed (typically, a request for \$OLDPASS when there is no \$OLDPASS).
59	Standard label violation.
60	Global RIN not available.
61	Group disc file space exceeded.
62	Account disc file space exceeded.
63	Non-sharable device (ND) capability required but not assigned.
64	Multiple RIN (MR) capability required but not assigned.
66	Plotter limit switch reached.
67	Paper tape error.
68	System internal error.
69	Miscellaneous (ATTACHIO) input/output error.
71	Too many files opened for process.
72	Invalid file number.
73	Bounds check violation.
77	NO-WAIT input/output operation is pending. (Series II only)
78	There is no NO-WAIT input/output for any file. (Series II only)
79	There is no NO-WAIT input/output for file specified. (Series II only)
80	Configured maximum number of spoolfile sectors would be exceeded by this output request.
81	No SPOOL class defined in system.
82	Insufficient space in SPOOL class to honor this input/output request.
83	Extent size exceeds maximum allowable.
84	The next extent in this spoolfile resides on a device which is unavailable to the system (i.e., the device is =DOWN).
85	Operation inconsistent with spooling; e.g., attempt to read hardware status.
86	Spool process internal error.
87	Offset to data is greater than 255 sectors.
89	Power failure.
90	The calling process requested exclusive access to a file to which another process has access.
91	The calling process requested access to a file to which another process has exclusive access.
92	Lockword violation.
93	Security violation.
94	Creator conflict in use of FRENAME intrinsic (user is not the creator).
95	"BROKEN" terminal read.
96	Miscellaneous disc input/output error (device may require HP Customer Engineer attention).

Code (Decimal)	Meaning
97	CONTROL Y processing requested but no CONTROL Y PIN exists.
98	Input/output read time has overflowed.
99	Magnetic tape error. Beginning of tape (BOT) found while requesting a backspace record (BSR) or a backspace file (BSF).
100	Duplicate file name in the system file directory.
101	Duplicate file name in the job temporary file directory.
102	Directory input/output error.
103	System directory overflow.
104	Job temporary directory overflow.
105	Illegal variable block structure.
106	Extent size exceeds maximum allowable.
107	Offset to data is greater than 255 sectors.
108	Inaccessible file due to a bad file label.
109	Illegal carriage control option.
110	The intrinsic attempted to save a system file in the job temporary file directory.



DEL STATUS WORD SETTINGS

APPENDIX

H

Table H-1 lists DEL procedures and shows the meaning of the values returned to the DEL status word.

Table H-1. DEL Status Word Settings

PROCEDURE	VALUE RETURNED TO STATUS WORD	MEANING
Forms Access Procedures		
OPENFORM	0	Operation successful.
	-1	Not a form file.
	>0	Error code from "FCHECK".
FINDFORM	0	Operation successful.
	-1	Form not in form file.
	>0	Error code from "FCHECK".
GETFORM	0	Operation successful.
	>0	Error code from "FCHECK".
NEXTEDIT	0	Operation successful.
	-1	Last edit specification has already been accessed.
	>0	Error code from "FCHECK".
CLOSEFORM	0	Operation successful.
	>0	Error code from "FCHECK".
Terminal Access Procedures		
OPENTERM	0	Operation successful.
	-1	Terminal not a 2640.
	>0	Error code from "FCHECK".
WRITETERM	0	Operation successful.
	>0	Error code from "FCHECK".
READTERM	0	Operation successful.
	>0	Error code from "FCHECK".
TERMSTATUS	0	Operation successful.
	>0	Error code from "FCHECK".
CLOSETERM	0	Operation successful.
	>0	Error code from "FCHECK".
Input Edit Procedures	0	Data passed edit.
	-1	Data failed edit.
High-Level Interface Procedures		
SHOWFORM	0	Operation successful.
	-1	Form cannot be located.
	>0, ≤ 999	Error code from "FCHECK" (WRITETERM or READTERM).
	>1000	Error code from "FCHECK" (FINDFORM or GETFORM).
EDITFIELD	0	Operation successful.
	-3	Required edit not one of DEL procedures.
	-2	Last edit specification has been accessed.
	-1	Failed edit check.
	>0	Error code from "FCHECK".

A

After Edit, Set Flag #, 3-4
 Alphabetic Data Field, 7-1
 Alphabetic Space-Filled Field, 7-2
 ALPHAEDIT, 7-1
 ALPHAFILL, 7-2
 Alphanumeric Data Field, 7-2
 ANEDIT, 7-2
 Applications of DEL, 1-2
 ASCII Character Set, D-1

B

BASIC Program, 9-12
 :BYE Command, C-17

C

Chain to Form Name, 3-4
 CLOSEFORM, 5-6
 CLOSETERM, 6-5
 Closing a Form File, 5-6
 Closing a Terminal File, 6-4
 COBOL Program, 9-1

D

Data Area Formats, A-1
 DEL Procedures, 4-1
 DEL Status Word Settings, H-2
 DEL/3000 Communications Area, A-1
 Deleting a Form File, 3-18
 Detecting/Correcting Errors, 9-19
 Directory Record Format, B-3
 Display Enhancement Features, 3-11

E

EDITFIELD, 8-3
 Editing Fields, 3-13, 8-2
 Edit Procedure Name, 3-4
 Edit
 Procedures, 7-1
 Summary, 7-6
 Edit Type, 3-4
 Editing Specification Sheet, 3-5, F-2
 Editing Specifications Tables, A-1
 ENHANCE DISPLAY key, 3-11
 Error Messages, 10-1

F

FCHECK Error Codes, G-1
 Features of DEL, 1-3
 Field Editing, 3-13
 Field Name, 3-4
 :FILE Command, C-7
 File ID Record Format, B-3
 FINDFORM, 5-3
 Form Creation Display, 3-10
 Form Definition Record Format, B-3
 Form File
 Closing, 5-6
 Format, B-1
 Opening, 5-1
 Form File Characteristics, 3-7
 Changing of, 3-9
 Form File Name, 3-1
 Form Layout Sheet, 3-3
 Sample, E-2
 Form Name, 3-2
 FORMAINIT
 Using, 3-6
 Terminating, 3-19
 Format of Form File, B-1
 Forms
 Changing, 3-16
 Creating, 3-10
 Deleting, 3-18
 Designing, 3-1
 Displaying, 3-17
 Listing, 3-15
 Locating, 5-3
 Moving to a Program Buffer, 5-4
 Function Selection Display, 3-7
 Function Selection Display Errors, 3-8

G

GETFORM, 5-4

H

:HELLO Command, C-3
 High-Level Interface Procedures, 8-1

I

Interface Procedures, 8-1
 Input/Edit Description Record Format, B-4
 Input/Output Files, 9-18

K

Keyboard Functions, 2-2

L

Length of Unprotected Field, 3-4

Locating a Form, 5-3

Look-Up Tables, 9-18

M

M11CREATE, 7-5

M11VERIFY, 7-5

Modulo 11 Check Digit Create, 7-5

Modulo 11 Check Digit Verify, 7-5

Moving a Form to a Program Buffer, 5-4

MPE Commands Specifications, C-1

N

NEXTEDIT, 5-5

NRANGE, 7-4

Numeric Data Field, 7-3

Numeric Range Edit, 7-4

Numeric Zero-Filled Field, 7-4

NUMRCEDIT, 7-3

O

OPENFORM, 5-1

Opening a Form File, 5-1

Opening a Terminal File, 6-1

OPENTERM, 6-1

P

Page Number, 3-4

Parameters, 4-5

Programs

Executing, 9-12

Preparing, 9-12

Writing, 9-1

R

Reading Input from a Terminal, 6-2

READTERM, 6-3

Requesting Terminal Status, 6-3

S

Sales Transaction Forms, 3-2, 3-12

SHOWFORM, 8-2

System Requirements, 1-5

T

Terminal File

Closing, 6-4

Opening, 6-1

Terminal Keyboard Functions, 2-1

Terminal Operation, 2-5

Terminal Requirements, 2-1

Terminating FORMAIN, 3-19

TERMSTATUS, 6-4

Test Flag # Before Edit, 3-4

W

WRITETERM, 6-2

Writing Output to a Terminal, 6-2

Writing Programs, 9-1

Z

ZEROFILL, 7-4

Part No. 30000-90050
Printed in U.S.A. 3/77

HEWLETT  PACKARD

Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd., Santa Clara, California 95050