HP 3000 Computer Systems

Data Entry Library

Reference Manual

HEWLETT hp PACKARD

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

Part No. 30000-90050
Product No. 32206A

Printed in U.S.A. 5/78

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.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

Third Edition . . . . May 1978

PRINTING HISTORY

New editions are complete revisions of the manual. 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 of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

The software product part number printed alongside the date indicates the version and update level of the software product at the time the manual edition or update was issued. Many product updates and fixes do not require manual changes, and conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	Jun 1976	32206A.00
Second Edition	Mar 1977	32206A.00
Third Edition	May 1978	32206A.01.07

PREFACE

This manual explains how to use the Data Entry Library (DEL/3000) for the HP 3000. (Operational differences are noted for MPE-C as required.) 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 264x terminals as described in the following documents:

Manual Title	Part No.
HP 3000 Series I MPE Commands Reference Manual	30000-90088
HP 3000 MPE Commands Reference Manual	30000-90009
HP 3000 Series I MPE Intrinsics Reference Manual	30000-90087
HP 3000 MPE Intrinsics Reference Manual	30000-90010
HP 3000 Series I System Manager/System Supervisor Reference Manual	30000-90089
HP 3000 System Manager/System Supervisor Reference Manual	30000-90014
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
2640B Display Station User's Manual	02640-90109
2641A APL Display Station User's Manual	02641-90001
2641A/2645A/2645S Display Station Reference Manual	02645-90005
2644A Mini DataStation Owner's Manual	02644-90001
2644A Mini DataStation Reference Card	5952-9950
2645A Display Station User's Manual	02645-90001
2645K Display Station User's Manual Supplement	02645-90030
2648A Graphics Terminal User's Manual	02648-90001

CONVENTIONS USED IN THIS MANUAL

NOTATION **DESCRIPTION**

[ ] An element inside brackets is *optional*. 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 *name*
name 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? $\underline{\text{ALPHA1}}$

superscript C Control characters are indicated by a superscript C

Example: Y^C

return *return* in italics indicates a carriage return

linefeed *linefeed* 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

Section II

Section III

Section IV

Section V

Section VI

Section VII

Section VIII

Section IX

Section X

Appendix A

Appendix B

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCING DATA ENTRY LIBRARY

SECTION I

The Data Entry Library (DEL/3000) helps you use an HP 264x terminal* to create and maintain files of formatted data on an HP 3000 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.

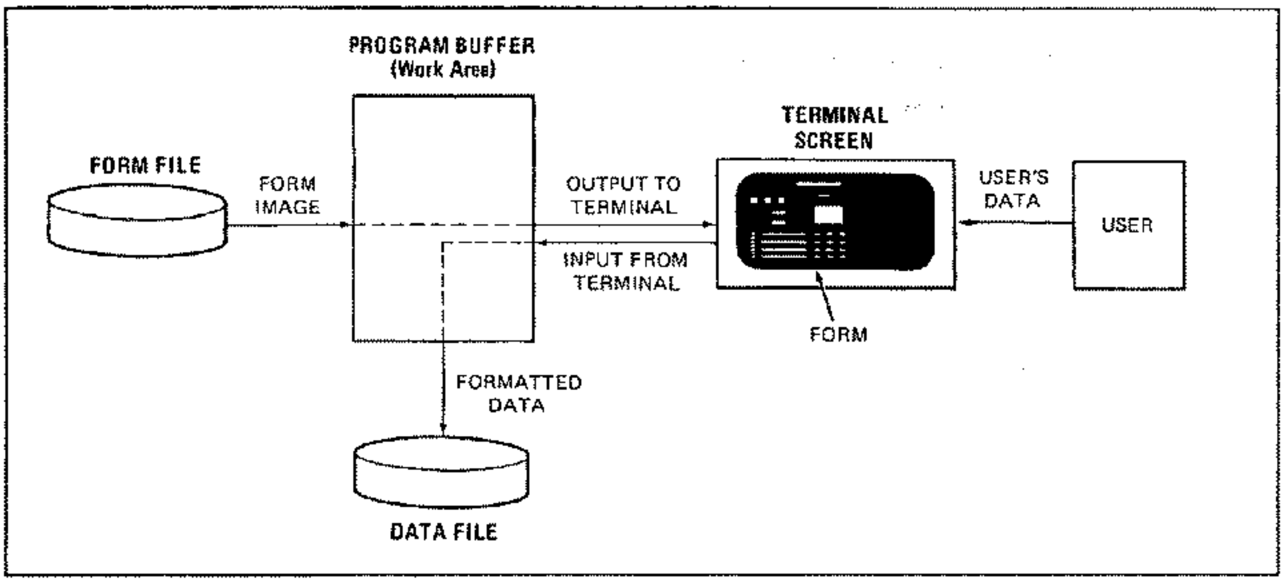


Figure 1-1. Application Program Data Flow

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.

*Note: Whenever the term HP 264x terminal or the word terminal appears in this manual, it applies to the HP 2640A, HP 2640B, HP 2641A, HP 2644A, HP 2645A, HP 2645K, and HP 2648A, but to no other terminals, unless otherwise noted.

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.

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. (Note that these procedures do not edit Katakana characters.)

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 264x terminals. It is easy to use, displaying convenient interactive prompts for form creation, modification, and selection of editing routines, and reducing 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 HP 264x 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.

TO COMPUTER SYSTEM
STEP 2 DESIGNING AND WRITING THE PROGRAM
STEP 4 TESTING AN APPLICATION

Figure 1-2. Using DEL in an Application

Step 1: Designing the Form

Analyze the requirements of your form by determining the permanent headings required, the data to be entered, and how these fields of information can best be arranged and presented. Then sketch the form on a Form Layout Sheet, indicating where each field is to begin and end.

Step 2: Creating the Form

Run the FORMAINT program to create your form in the system. You may store the form in an existing form file or create a new file for this purpose. To define the form, transcribe your entries from the Form Layout Sheet onto the terminal screen via the keyboard. If you make any mistakes, you simply correct them as you go along, or re-run FORMAINT later for this purpose.

Step 3: Designing and Writing the Data Entry Program

Write a program to interface with the person who will enter data into the formatted data file and keep it updated. The possible structure and details of such programs are wide and varied. Examples appear later in this manual.

Step 4: Testing the Application

Once you have created the forms and programs required, your normal application program validation procedures apply.

Step 5: Running the Program for Data Entry

Your application is ready for data entry use.

SYSTEM REQUIREMENTS

DEL operates on any HP 3000 Computer System with at least one HP 264x terminal. DEL runs under control of MPE, and can be used with the COBOL, FORTRAN and SPL compilers and the BASIC interpreter. No other hardware or software is needed.

USING THE TERMINAL

SECTION II

If you have not yet operated an HP 264x 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 2640B Display Station User's Manual*
- *HP 2641A APL Display Station User's Manual*
- *HP 2644A Mini DataStation Owner's Manual*
- *HP 2645A Display Station User's Manual*
- *HP 2645K Display Station User's Manual Supplement*
- *HP 2648A Graphics Terminal User's Manual*

TERMINAL REQUIREMENTS

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

- Equipped for line/FORMAT or page/FORMAT mode. When operating in FORMAT 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.) BLOCK MODE/PAGE is supported on all terminals except the HP2640A and BLOCK MODE/LINE is supported on all 264x terminals. Only BLOCK MODE/LINE is supported when using DEL with MPE-C.
- Connected to the computer over an asynchronous terminal controller.

PAGE AND LINE MODES. DEL operates in both BLOCK MODE/PAGE and BLOCK MODE/LINE depending on the internal configuration of the terminal being used. DEL operates only in BLOCK MODE/LINE when used with MPE-C. To use BLOCK MODE/PAGE, the D and G strapping options on the keyboard interface printed circuit board (which is labeled KEYBD I/F) inside the terminal must be open (strap pulled out). In addition, the F strapping option must be open on 2640B terminals. Although the HP 2640A has these straps, the G strap has no effect. Therefore, BLOCK MODE/PAGE is not supported on the HP 2640A. It is supported on all other terminals.

The HP 2641A, 2645A, 2645K, and 2648A terminals have the capability to have the strapping options and latching keys set programmatically. DEL makes use of this feature by setting the terminal in BLOCK MODE/PAGE mode when the terminal is opened with the OPENTERM procedure (refer to Section VI). You may suppress this feature by setting a flag in the Terminal Mode Information word in the communication area (refer to Appendix A). If the automatic configuration is used, the normal modes are restored when the terminal is closed using the CLOSETERM procedure.

Note that FORMAINT programmatically overrides the physical strapping (except when operating with multipoint) to ensure that it operates in BLOCK MODE/LINE mode.

Multipoint terminals may not operate in BLOCK MODE/LINE mode. (See below.)

During BLOCK MODE/PAGE operation, echo is always off, and the input terminators are DC2 (whose function is a block transfer enable from the terminal) and RS (a record separator). These ASCII characters replace CR (carriage return) as the input terminators. These codes are automatically generated by the terminal in block mode. If you are debugging a program with the MPE DEBUG intrinsic and the terminal does not respond to a carriage return, or if you have pressed the BREAK key to interrupt execution of a program that calls DEL procedures, try entering CONTROL R (R^c which is ASCII DC2) to terminate your input. To reset echo-on after breaking a program, enter ESC: (Escape colon), or switch the terminal to half duplex until you are ready to resume execution of the program. Before entering the :RESUME command, restore echo-off by entering ESC;(Escape semicolon) and be sure the terminal switch is set to full duplex.

When there are many terminals simultaneously transmitting data in BLOCK MODE/PAGE, there is a small chance that a data overrun error could occur. DEL can detect this when it occurs. Refer to the READTERM procedure discussion for an explanation of how this situation is handled. When using BLOCK MODE/PAGE, it is recommended that the number of MPE terminal buffers (TBUF) be at least 128. The number of buffers is specified by the system supervisor at system configuration time. (Refer to the *System Manager/System Supervisor Reference Manual.)*

NOTE

> The MPE READ intrinsic which is used by the COBOL ACCEPT statement cannot be used in BLOCK MODE with a terminal that is strapped for BLOCK MODE/PAGE operation. This is due to the input termination characters used by the terminal in BLOCK MODE/PAGE.

MULTIPOINT TERMINALS. Multipoint terminals may be used with DEL. The following characteristics of DEL multipoint operation should be noted:

- Multipoint terminals are always MPE terminal type 14.
- Multipoint terminals operate in BLOCK MODE/PAGE only. A negative value in the Terminal Information Word of the communications area (refer to Appendix A) has no effect.
- Multipoint terminals require a portion of the terminal memory for the Data Communications buffer. Therefore, care should be taken that the size of a form does not exceed the available terminal memory.
- The design of multipoint terminals restricts an individual terminal read to a maximum of 2048 characters. Therefore, forms containing more than 2048 displayable and non-displayable characters cannot be created or modified on a multipoint terminal. Application programs reading data from forms are limited to 2048 characters of data. (Refer to the description of the READTERM procedure for more information.)

TERMINAL BUFFERS. If it is anticipated that many users will be using FORMAINT or the DEL procedures simultaneously, it is recommended that the system supervisor configure the system with the maximum number of terminal buffers (255). Refer to the *System Manager/System Supervisor Reference Manual* for more information about how to do this.

TERMINAL KEYBOARD FUNCTIONS

The HP 264x terminal keyboard provides 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 264x 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, *ESC U* 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 2641) or J^c (on all terminals)	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 *return*) entered or received. With 128-character Roman character set, displays escape codes and control functions

Table 2-1. Keyboard Functions (Continued)

<table>
<tr><th>KEY GROUP</th><th>KEY</th><th>FUNCTION</th></tr>
<tr><td></td><td></td><td>associated with an entry as an aid in debugging. (See owner's manual.)</td></tr>
<tr><td></td><td>BLOCK MODE</td><td>When latched down, places terminal in <i>block mode.</i></td></tr>
<tr><td></td><td>REMOTE</td><td>When latched down, places terminal in remote (on-line) operating mode for communicating with the computer.</td></tr>
<tr><td></td><td>TAB</td><td>(In FORMAT mode), moves cursor to start of next unprotected field; disregards normal horizontal tab stops.</td></tr>
<tr><td></td><td>ENTER</td><td>In DEL's FORMAINT 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.</td></tr>
<tr><td></td><td>ENHANCE DISPLAY (on HP 2640)

f1^c (on all other terminals)

NOTE

The superscript c indicates the CNTRL function – thus, you must hold down the CNTL key while pressing f1.</td><td>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>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><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 other terminals 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.</td></tr>
<tr><td></td><td>START UNPROTECTED FIELD (on HP 2640)

f2^c (on all other terminals)</td><td>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.)</td></tr>
</table>

Table 2-1. Keyboard Functions (Continued)

KEY GROUP	KEY	FUNCTION
	END UNPROTECTED FIELD (on HP 2640) f3^{c} (on all other terminals)	Ends an unprotected field. Characters from present cursor position to end of this line or start of *next* unprotected field, are protected in FORMAT mode and cannot be overwritten.
	FORMAT MODE (on HP 2640) f4^{c} (on all other terminals)	Places terminal in FORMAT mode (ON) so that only *unprotected fields* can be altered by either terminal user or computer. (The unprotected fields are *specifically defined* 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 other terminals, 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 all other terminals)	Special function key that returns you to Function Selection Display in FORMAINT. 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.
	R^{c} (Control R) (ASCII DC2)	Acts as end-of-data indicator to terminate input when in BREAK or DEBUG and BLOCK MODE/PAGE is set.

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.*

The terminal type is set to 10 by the DEL OPENTERM procedure unless it is already 10, or it is a multipoint terminal (type 14) or a Katakana terminal (type 12). The log-on terminal type is restored by CLOSETERM. If you log on to an MPE-C system, it is recommended that you specify terminal type 10 (using the TERM= parameter of the :HELLO command). The terminal type cannot be altered programmatically in these systems.

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. (To do a hard reset on an HP 2641, 2645, or 2648, press the RESET TERMINAL key twice.

2. Set the A.C. POWER switch to the OFF position, turning off the power.

NOTE

> It is recommended that you do not use the memory lock feature of the 264x terminals when using DEL. If the cursor is positioned in the locked area of the terminal when a read is executed, a read error will occur.

DESIGNING AND ENTERING FORMS

SECTION III

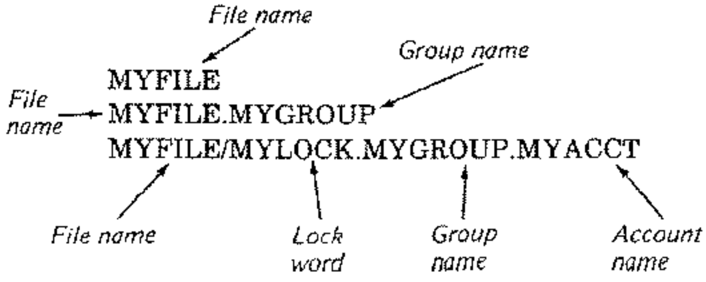
The first steps in any DEL/3000 application involve designing and entering the form (or forms) that the application will use. These steps and those that follow are best illustrated by an example. Suppose, for instance, that you are preparing an application that enables a clerk in a construction supply 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 } (Six-digit numeric code)
- Salesman identification number
- 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:



```
MYFILE
MYFILE.MYGROUP
MYFILE/MYLOCK.MYGROUP.MYACCT
```

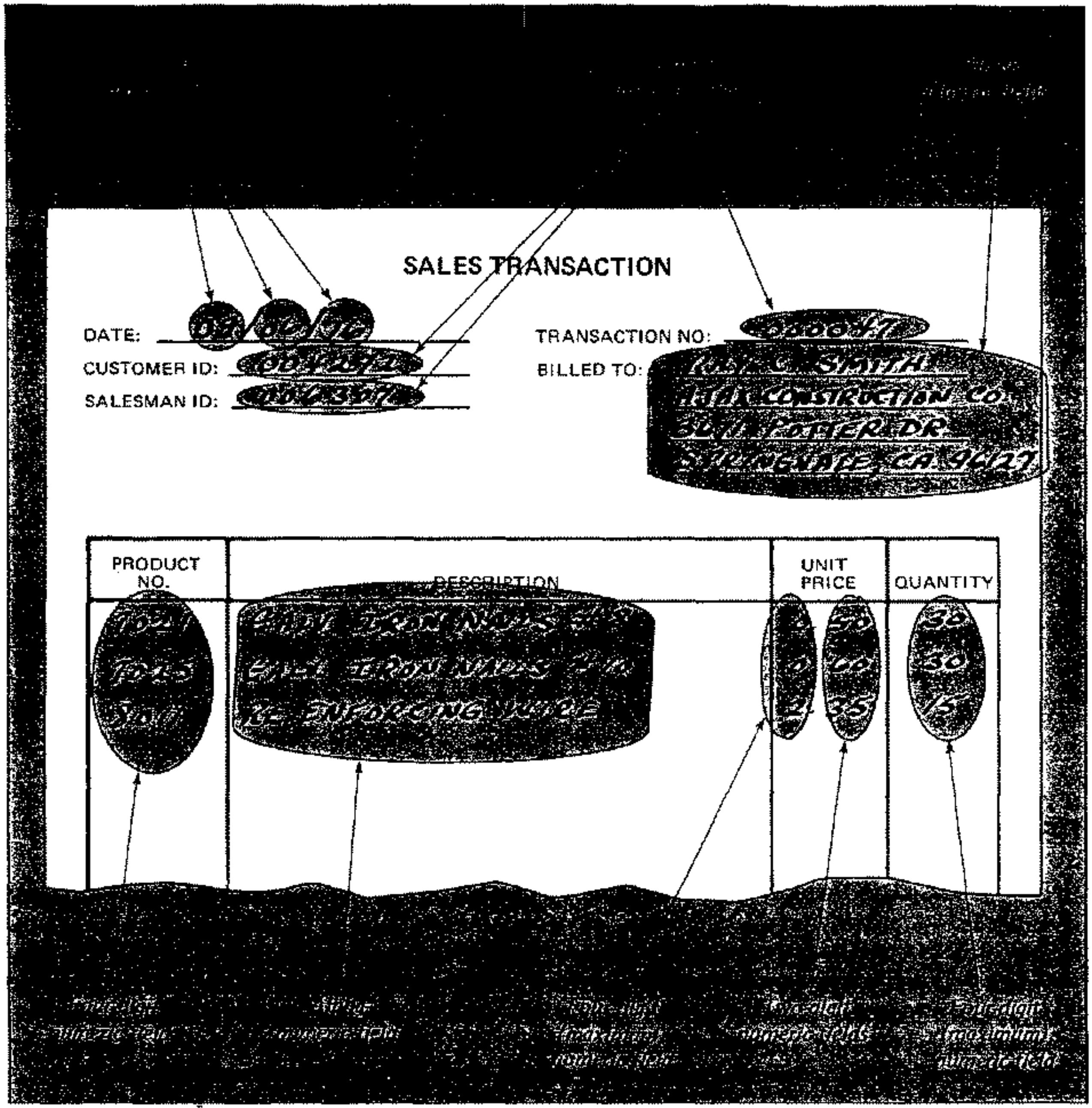


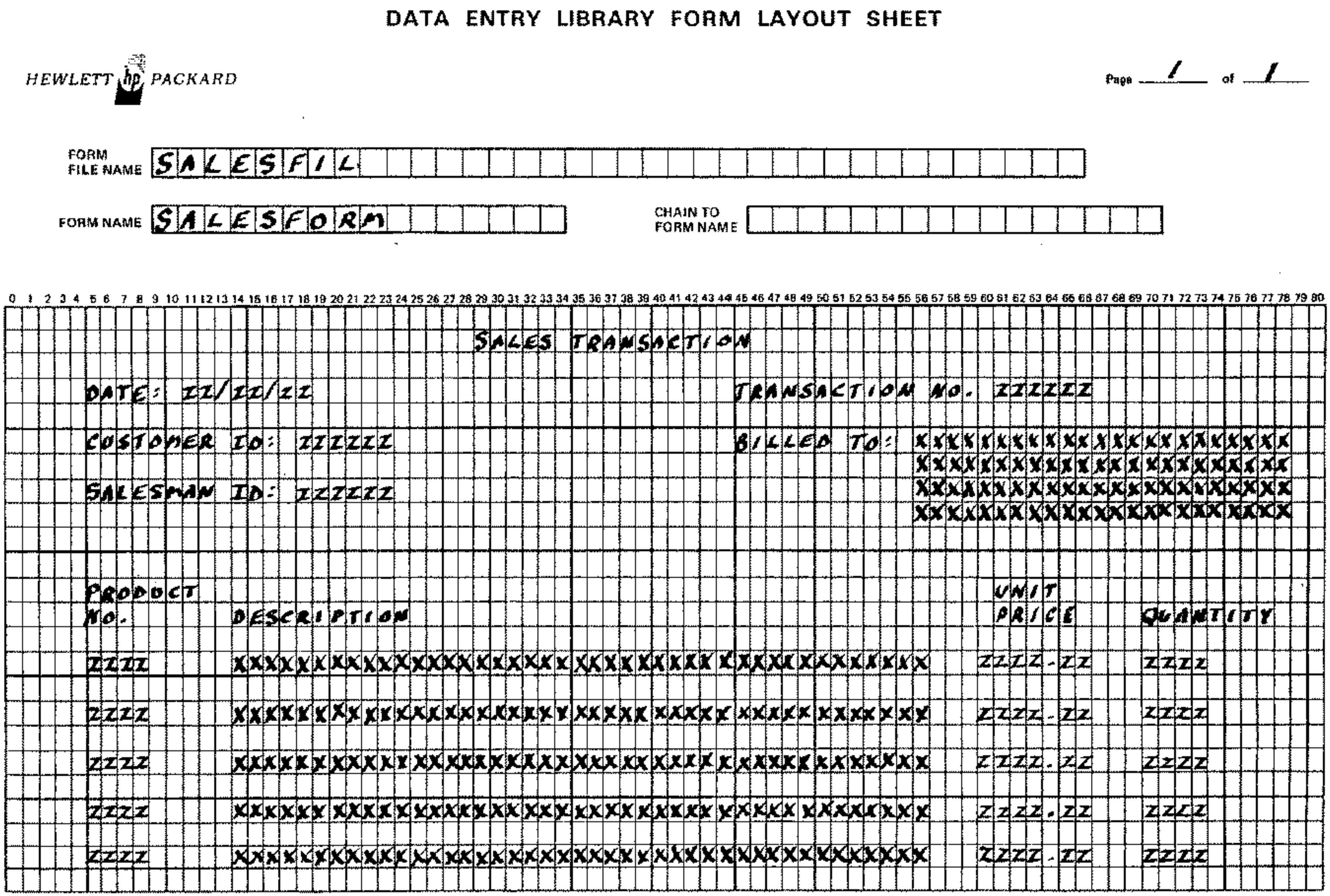
Figure 3-1. Sales Transaction Form

On the Form Layout Sheet for the sales transaction report, the form file name SALESFIL is used.

- *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.



DATA ENTRY LIBRARY FORM LAYOUT SHEET

HEWLETT PACKARD

Page 1 of 1

FORM FILE NAME SALESFIL

FORM NAME SALESFORM

CHAIN TO FORM NAME

```
                            SALES TRANSACTION

     DATE: ZZ/ZZ/ZZ                         TRANSACTION NO. ZZZZZZ

     CUSTOMER ID: ZZZZZZ                    BILLED TO: XXXXXXXXXXXXXXXXXXXXXXXX
                                                       XXXXXXXXXXXXXXXXXXXXXXXX
     SALESMAN ID: ZZZZZZ                               XXXXXXXXXXXXXXXXXXXXXXXX
                                                       XXXXXXXXXXXXXXXXXXXXXXXX


     PRODUCT                                                UNIT
     NO.      DESCRIPTION                                   PRICE    QUANTITY

     ZZZZ     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   ZZZZ.ZZ   ZZZZ

     ZZZZ     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   ZZZZ.ZZ   ZZZZ

     ZZZZ     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   ZZZZ.ZZ   ZZZZ

     ZZZZ     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   ZZZZ.ZZ   ZZZZ

     ZZZZ     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   ZZZZ.ZZ   ZZZZ
```

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.

Forms containing more than 2048 displayable/non-displayable characters cannot be created or modified on multipoint terminals. However, you may display a form with as many as 4096 characters. The maximum number of displayable/non-displayable characters allowed on each line of a form is 600.

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 FORMAINT, 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. (Editing specifications should not be entered if the data is to be entered in Katakana characters.) 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.

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
								Low Value	High Value		
DATE (MONTH)	2	NUMERIC	CNRANGE								
		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	CNUMRCEDIT								
CUSTOMER I.D.	6	NUMERIC	CNUMRCEDIT								
SALESMAN I.D.	6	NUMERIC	CNUMRCEDIT								
PRODUCT NO.	4	NUMERIC	CNUMRCEDIT								
UNIT PRICE	4	NUMERIC,	CZEROFILL								
(DOLLARS)		ZERO-FILL									
UNIT PRICE	4	NUMERIC,	CZEROFILL								
(CENTS)		ZERO-FILL									
QUANTITY	4	NUMERIC	CNUMRCEDIT								

Figure 3-3. Transaction Form Editing Specification Sheet

- *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.
- *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 FORMAINT program by logging-on and entering:

```
:RUN FORMAINT.PUB.SYS
```

If you have logged-on at a terminal other than an HP 264x, FORMAINT prints the following message and immediately terminates:

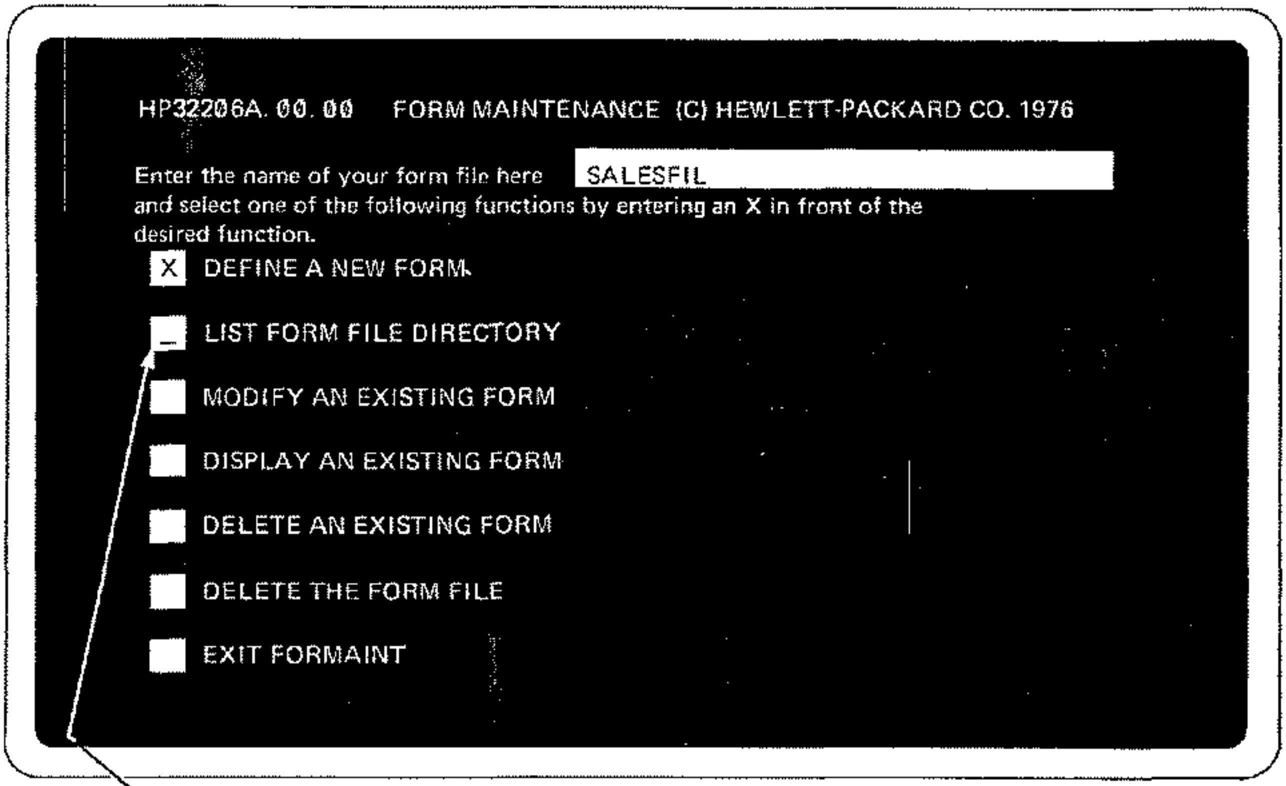
```
INPUT DEVICE IS NOT A 264x
```

If you have logged-on to an HP 264x terminal that cannot be set to block mode programmatically, FORMAINT prints the message:

```
PLEASE PRESS BLOCK MODE KEY AND AUTO LF.
```

Latch the BLOCK MODE key down, placing the terminal in block mode. FORMAINT 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. If the file does not exist, FORMAINT will create the form file as a permanent file on disc with the characteristics listed in table 3-1, when the entire screen contents are transmitted to FORMAINT as directed below. If the file name indicates an old (existing) file, FORMAINT will access that file. In the example in figure 3-4, the name of the new file SALESFIL (indicated on the layout sheet) is entered.



```
HP32206A.00.00    FORM MAINTENANCE (C) HEWLETT-PACKARD CO. 1976

Enter the name of your form file here  SALESFIL
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 FORMAINT
```

Cursor position

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

*only 1000 records are allocated until more are needed.

NOTE

Because FORMAINT grants a user *exclusive access* to a form file, two or more users cannot use FORMAINT 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 FORMAINT 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 case only) in the input field.

3. Press the ENTER key to transmit your entries to FORMAINT 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, FORMAINT waits for you to correct the error and press ENTER again.

NOTE

> Several errors that you may encounter when using FORMAINT 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*	FORMAINT cannot open the form file you specified. In this message, *nnnn* indicates an error code return through FORMAINT by the FCHECK intrinsic, as discussed in *MPE Intrinsics Reference Manual* 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.
FORM FILE ACCESS ERROR CODE = *error number*	FORMAINT encountered a read or write error when accessing your form file. Refer to Appendix G for the meaning of the *error number* listed under the FCHECK intrinsic errors.
FORMAINT MUST HAVE UPDATE ACCESS TO FORMS FILE	The form file was not created to allow read/write access (ACC=INOUT).

RULES FOR USING FORMAINT DISPLAYS

When responding to the prompts for any FORMAINT display, remember these general rules:

- Press the TAB key to skip to a new prompt (input field).
- Press the ENTER key to indicate that you are finished with a display and continue with the normal progression of a function.
- Press the f8 key (or the f8 key with CNTL depressed on the HP 2640) to exit from any function and return to the Function Selection Display.
- Reposition the cursor to the beginning of the line and use the DELETE LINE key to correct an 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.

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;REC=64,16,F,ASCII;DISC=6000;SAVE
```

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 1000 directory entries and contains a maximum of 100 forms. Since the form file size can not be expanded, it is recommended that you estimate the maximum number of forms you intend to define and create a form file large enough to contain them.
- 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.

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.

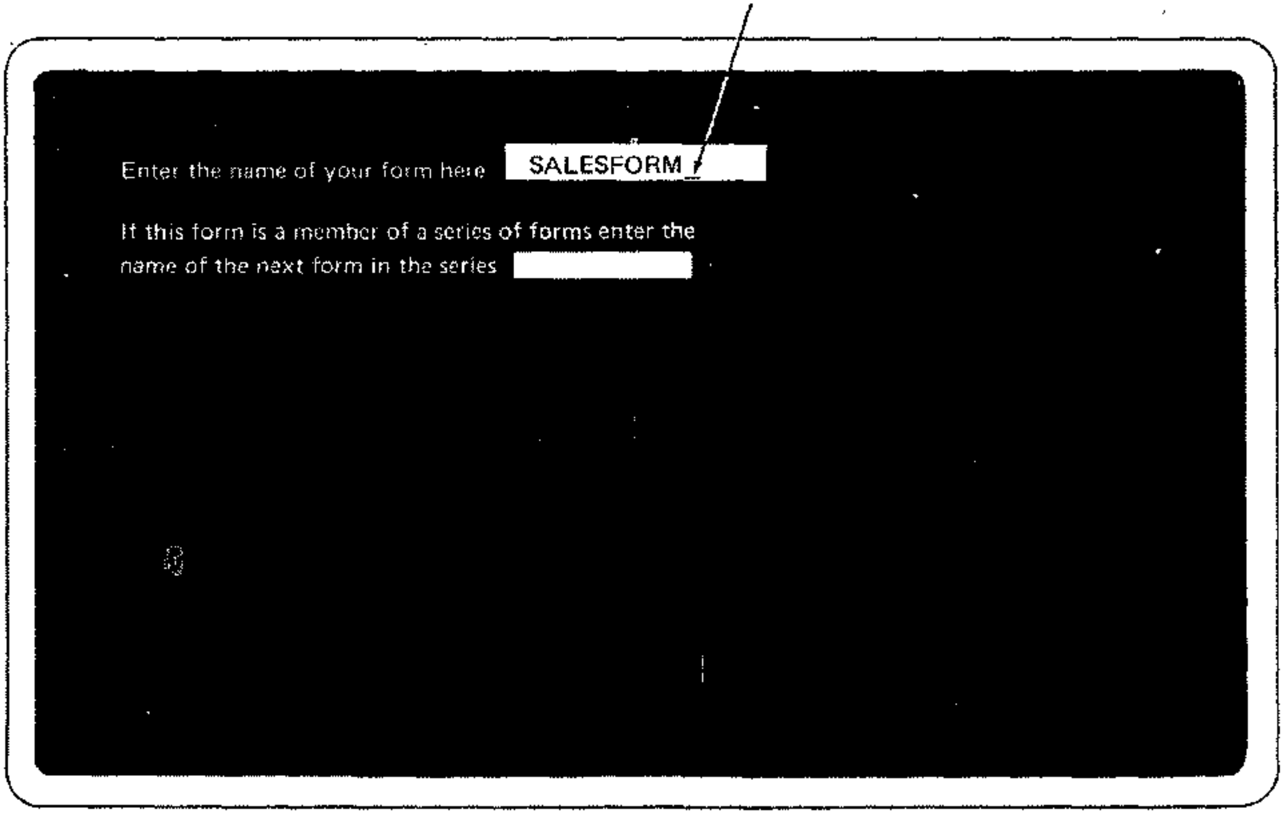
Respond to the first prompt in this display by entering the name you wish to assign your new form. You may copy this from your Form Layout Sheet. No other form of this name is permitted in this form file. The name SALESFORM is entered in response to this prompt in figure 3-5.

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).



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. FORMAINT returns to the Function Selection Display, allowing you to specify a new form file or function.
FORM ALREADY EXISTS IN *filename*	You specified a form that already exists in the form file named *filename.* FORMAINT waits for you to enter a new form name.
FORM NAME REQUIRED	You failed to enter a form name. FORMAINT 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 the terminal is placed in Format Mode all character positions on the screen are protected except those fields that have been specifically defined as unprotected or transmit only. (Transmit only fields are available only on HP 2641, 2645, and 2648 terminals.) For a definition of these types of fields refer to the *2641A/2645A/2648A Display Station Reference Manual.*

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 other terminals) 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 other terminals) to terminate the field.

NOTE

> FORMAINT 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, FORMAINT converts it into two separate fields.

2. To use the display enhancement features with an unprotected field:
   a. Press the ENHANCE DISPLAY key (on the HP 2640) or enter f1^c (on other terminals, 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. Press the START UNPROTECTED FIELD key (on HP 2640) or enter f2^c (on other terminals to begin the field.
   c. Enter the characters in the field. (Blanks are usually used.)
   d. Press the END UNPROTECTED FIELD key (on HP 2640) or enter f3^c (on other terminals) to terminate the field.
   e. Press the ENHANCE DISPLAY key (on the HP 2640) or enter f1^c (on other terminals), 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 define a transmit only field:

   a. Position the cursor to the beginning of the transmit only field.

   b. Press the ESC (Escape) key and left brace ({).

   c. Move the cursor to the column following the end of the transmit only field being defined.

   d. Press the ESC (Escape) key and right bracket (]).

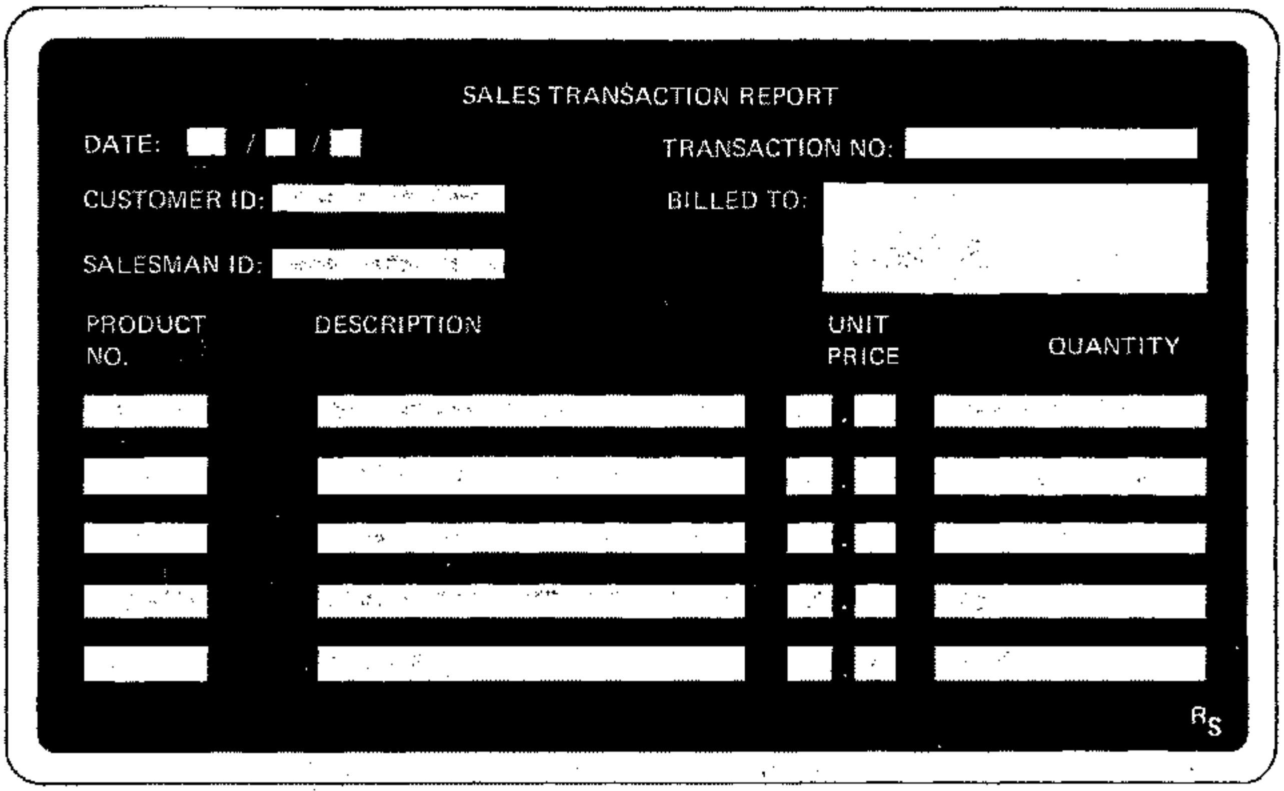
4. To indicate that the form is complete, enter the ^ (CONTROL ^) key, producing a record-separator. This indicates the end of the form and, on most terminals, displays the following characters on your screen:

   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.

5. 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: / /

TRANSACTION NO:

CUSTOMER ID:

BILLED TO:

SALESMAN ID:

PRODUCT NO.	DESCRIPTION	UNIT PRICE	QUANTITY
		.	
		.	
		.	
		.	
		.	

$^{R}{}_{S}$

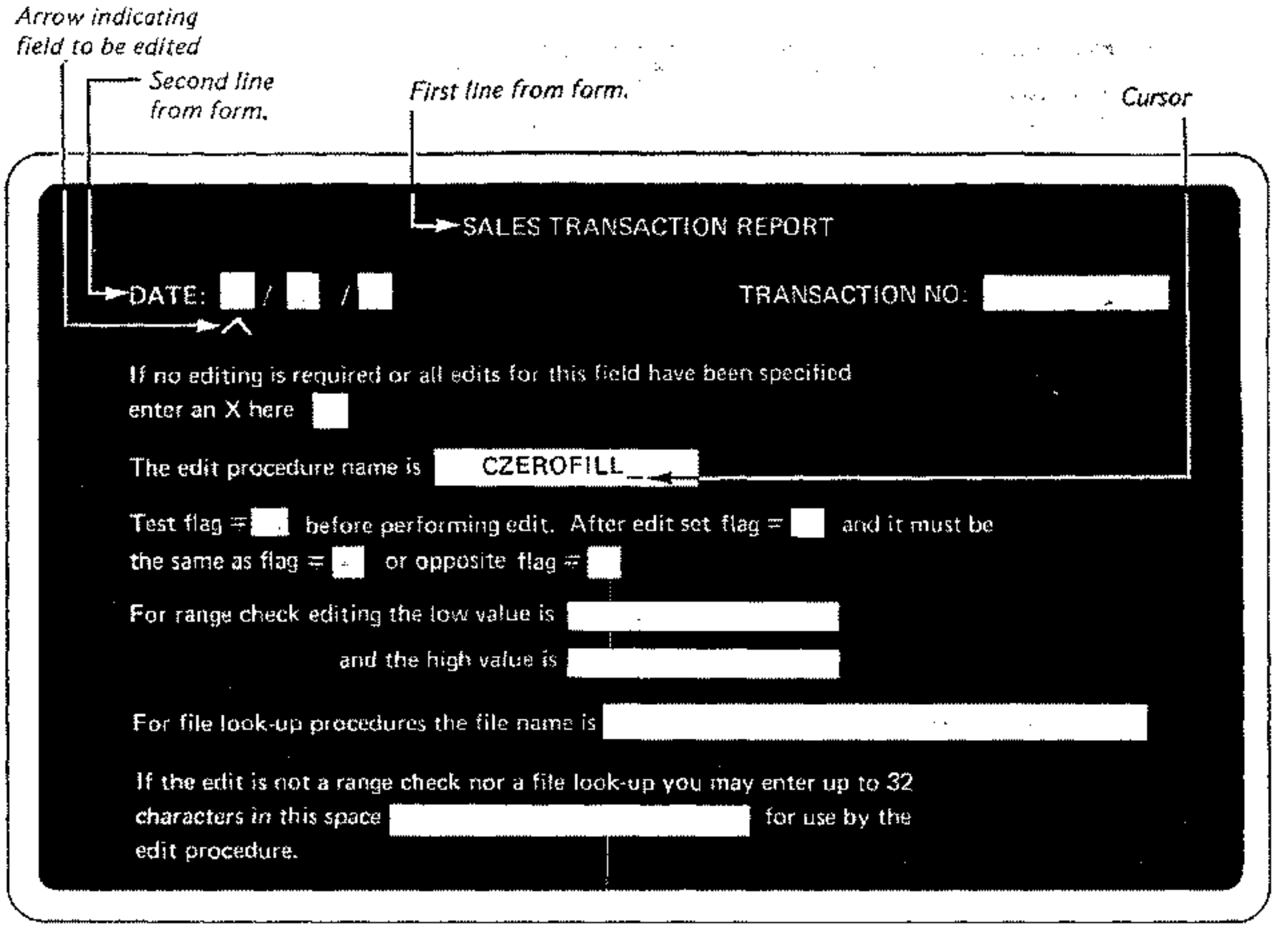
Figure 3-6. Sales Transaction Report Form As Entered

SPECIFYING FIELD EDITING. When the form is stored in the form file, FORMAINT 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, FORMAINT 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 terminal (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 FORMAINT, 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



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 (enter f8^c 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, FORMAINT 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 FORMAINT. 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, FORMAINT detects any erroneous entries and reports them as indicated in table 3-4.

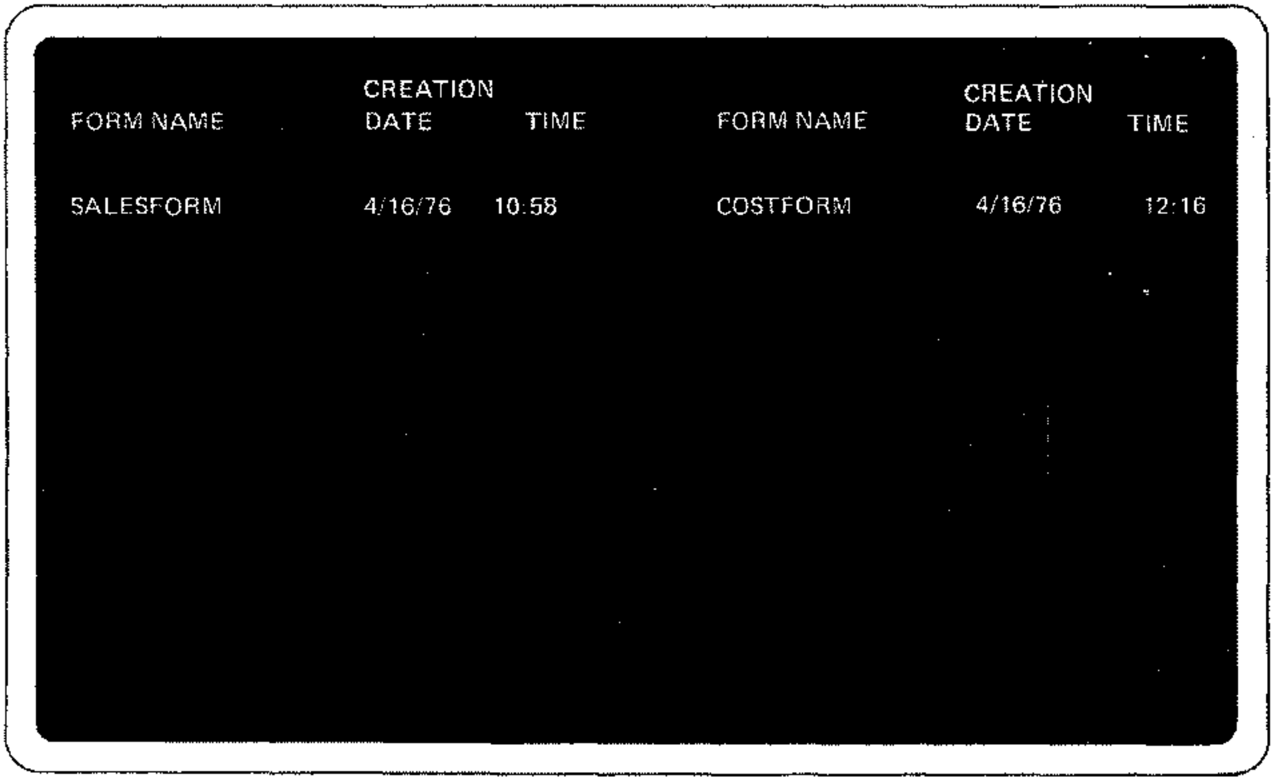
Table 3-4. Editing Specification Display Errors

MESSAGE	CAUSE
FORM DEFINITION EXCEEDS SYSTEM CAPABILITY (MPE-C message)	You defined a form which contains more than 216 characters on one line. This exceeds MPE-C system capability. (If the form is defined on a non-MPE-C System, however, it will be acceptable to MPE-C.)
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. FORMAINT causes the invalid entry to blink and waits for you to correct the error.
INVALID ROW ADDRESS	Form file may be invalid. Check form with DISPLAY function. If error, delete and reenter form.
SELECT ONE – RANGE CHECK, FILE LOOK–UP OR EDIT DEFINED DATA	You specified more than one of the following entries for an input field: *Range Check, Look-Up Procedure File Name,* or *Characters* (for edit-defined data). FORMAINT waits for you to correct the error.

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 FORMAINT 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 terminal (or $f8^c$ on the HP 2640).



```
                         CREATION                                         CREATION
   FORM NAME             DATE        TIME          FORM NAME              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. FORMAINT 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.

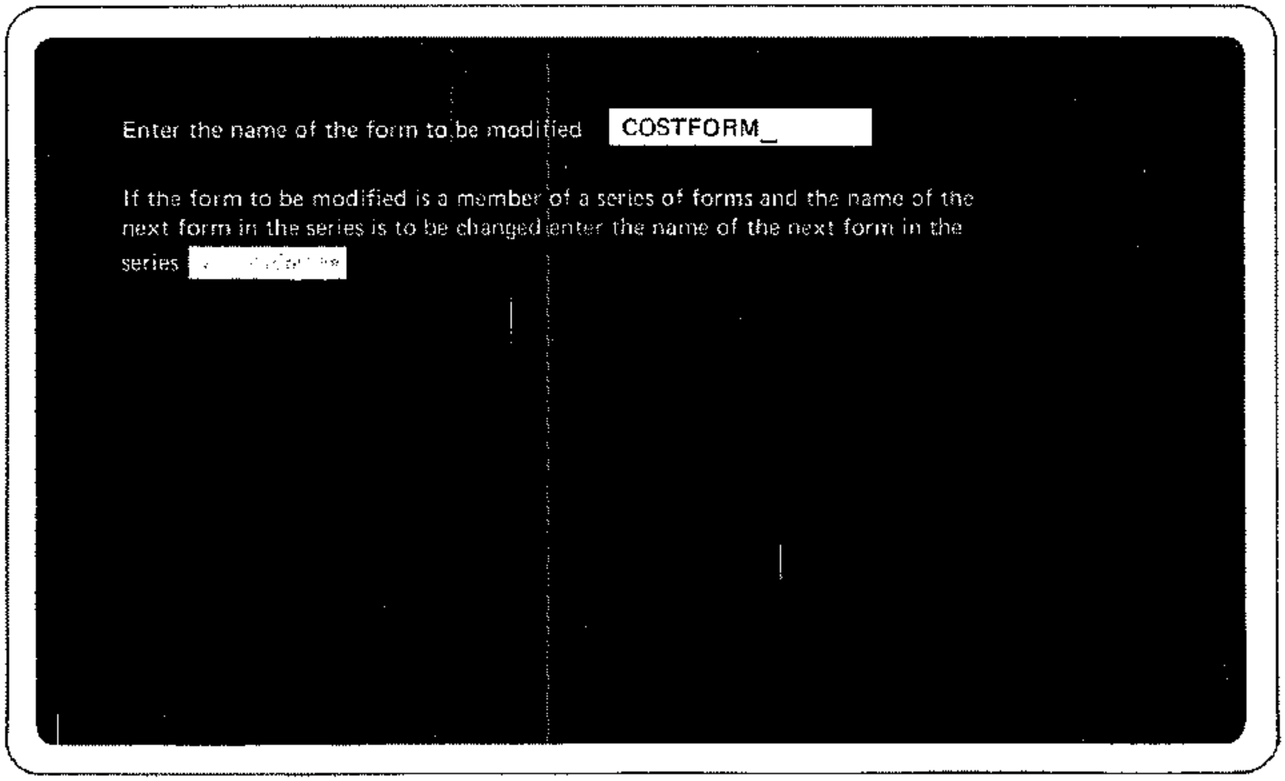
Transmit your responses to FORMAINT by pressing ENTER. FORMAINT then clears the screen and displays the form to be modified, allowing you to alter any entries you desire and to create new ones. Use the same methods you employed when first creating the form.

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 FORMAINT waits for you to enter a correct form name:

```
FORM DOES NOT EXIST IN filename
```



Enter the name of the form to be modified COSTFORM_

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.

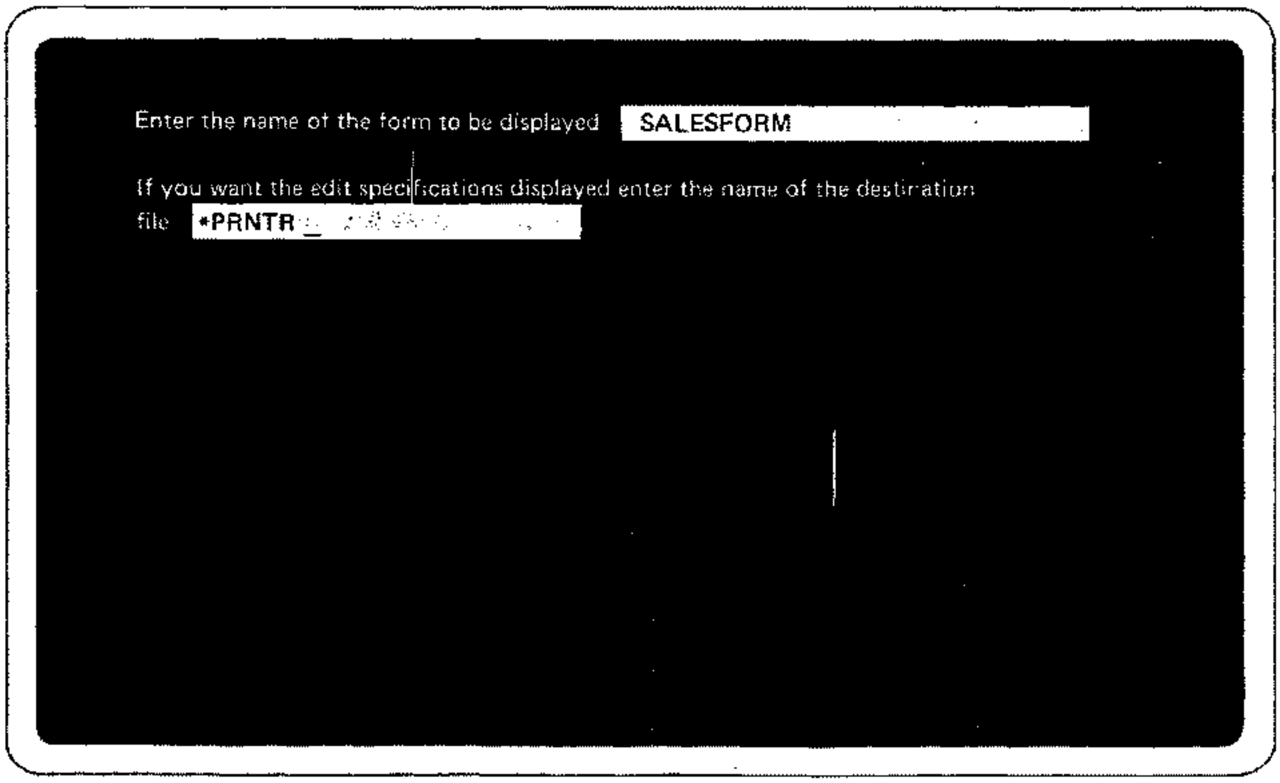
DISPLAYING A FORM

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
:RUN FORMAINT.PUB.SYS
```

File name of output file.

Device class name of output device (a printer).



Enter the name of the form to be displayed SALESFORM

If you want the edit specifications displayed enter the name of the destination file *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 FORMAINT.

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 (or f8^{c}on 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, FORMAINT 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, FORMAINT 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 FORMAINT 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, FORMAINT 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

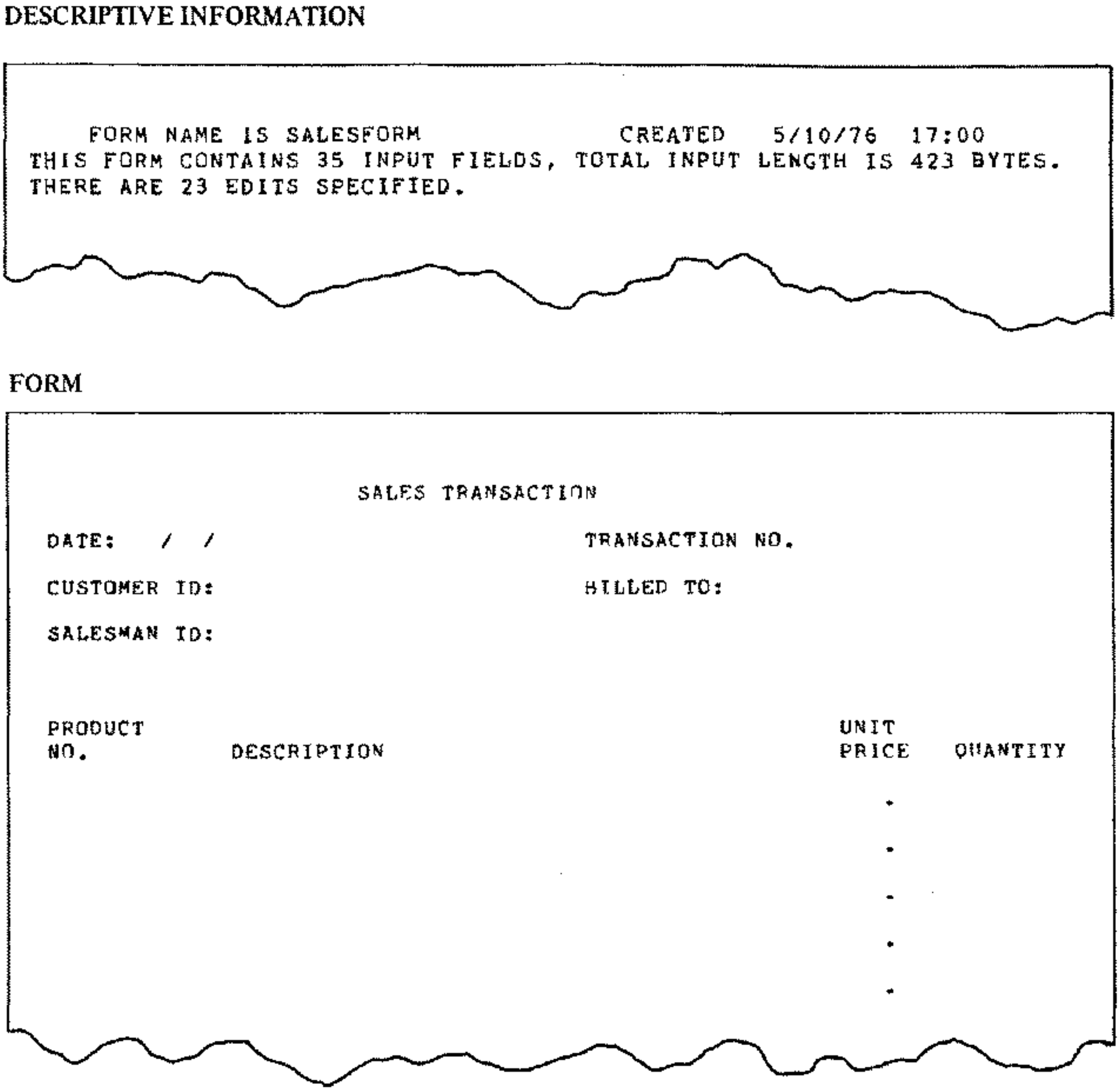
To delete a form from a form file, enter X at the DELETE AN EXISTING FORM prompt in the Function Selection Display and press ENTER. FORMAINT presents the *Form Deletion Display* (figure 3-12). Enter the form name at the prompt and press ENTER. The form is removed from the system immediately. If, however,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

FORM DOES NOT EXIST IN *filename*

When the form is deleted, FORMAINT 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, FORMAINT 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, FORMAINT 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                                                    UNIT
NO.          DESCRIPTION                                   PRICE   QUANTITY
                                                             .

                                                             .

                                                             .

                                                             .

                                                             .
```

Figure 3-11. Form Listing Output

TERMINATING FORMAINT

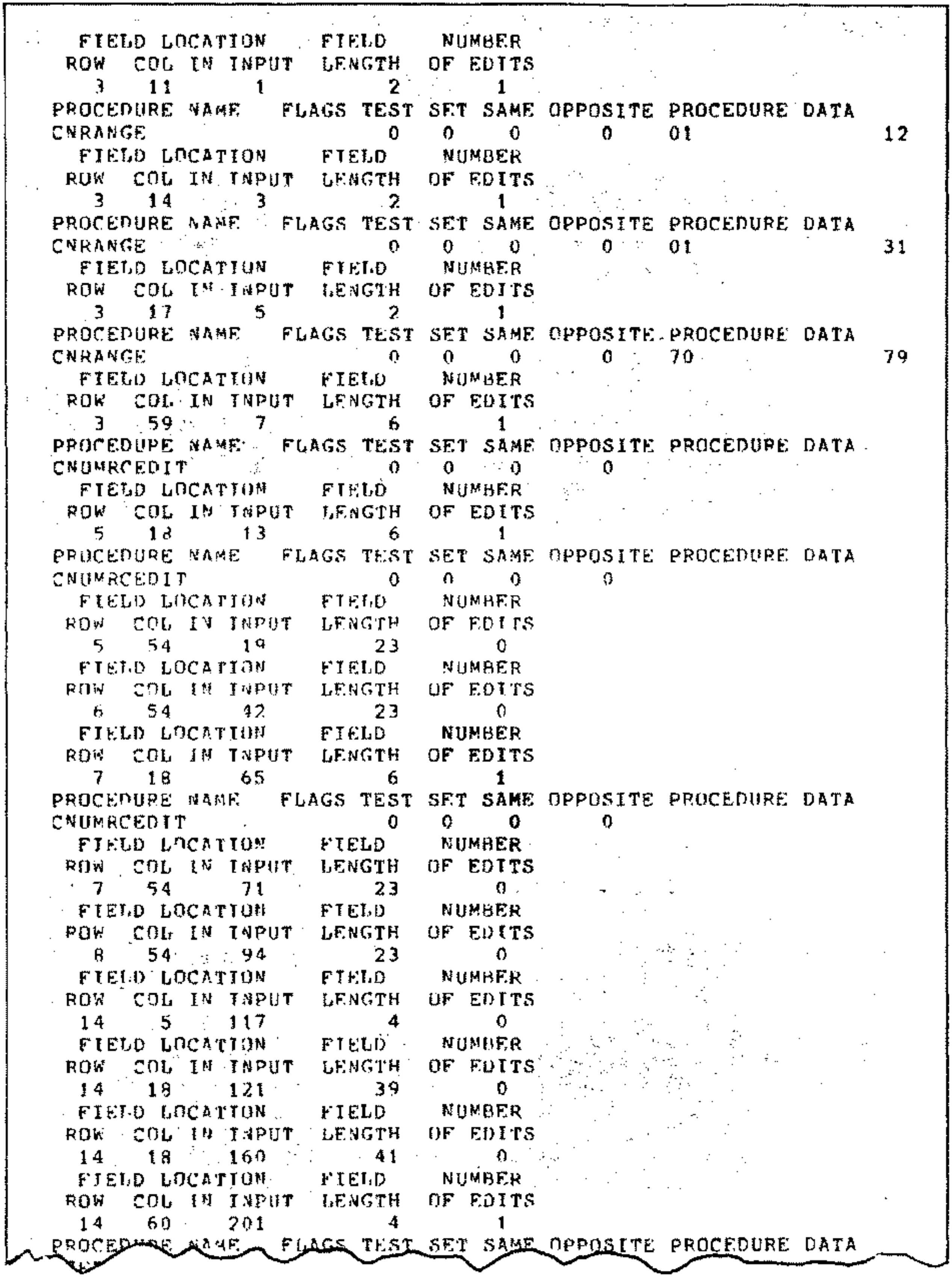
To terminate FORMAINT, closing all files accessed by it, enter X at the EXIT FORMAINT prompt in the Function Selection Display, and press ENTER. If block mode was not set programmatically, FORMAINT prints the message:

REMEMBER TO UNLATCH THE BLOCK MODE KEY.

In this case, remove the terminal from block mode by releasing the BLOCK MODE key. MPE resumes control, printing the message shown below and prompting you for a new command:

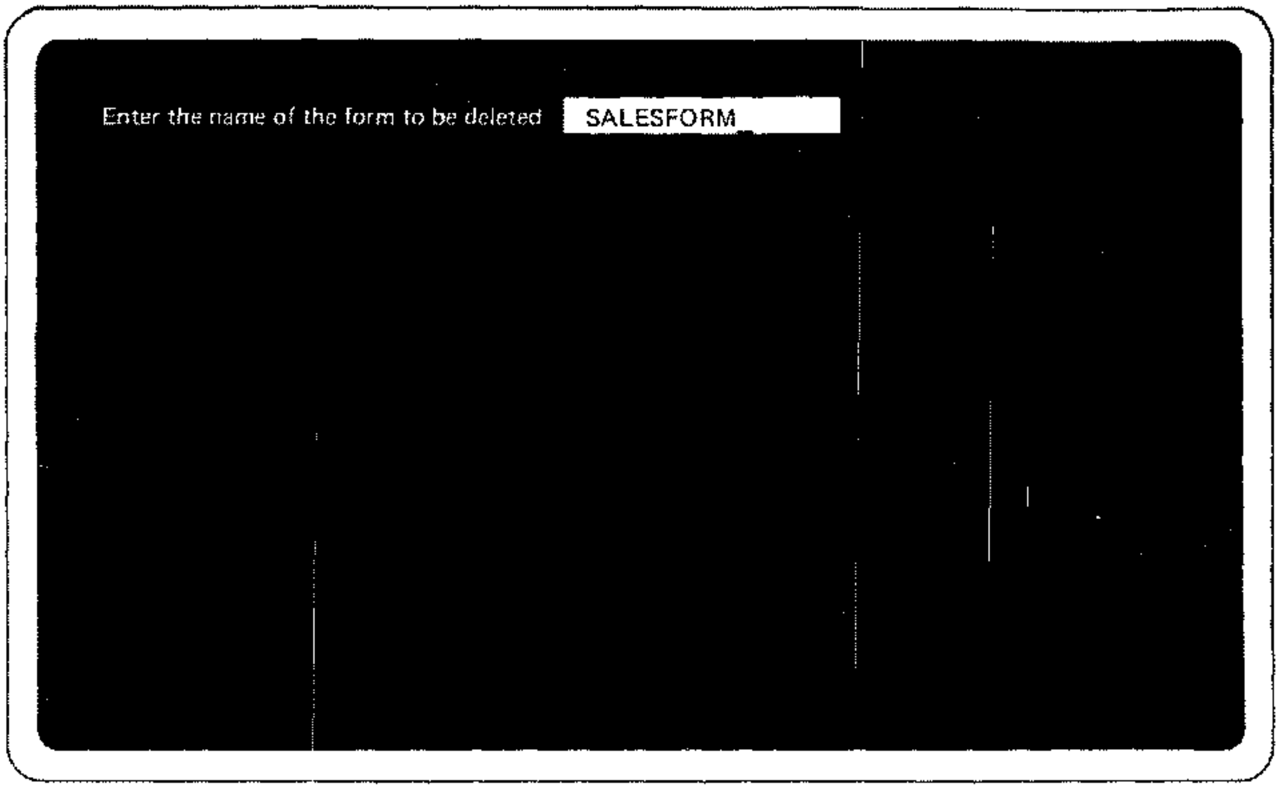
```
END OF PROGRAM
:
```

prompt for new MPE command.



```
  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      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      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
```

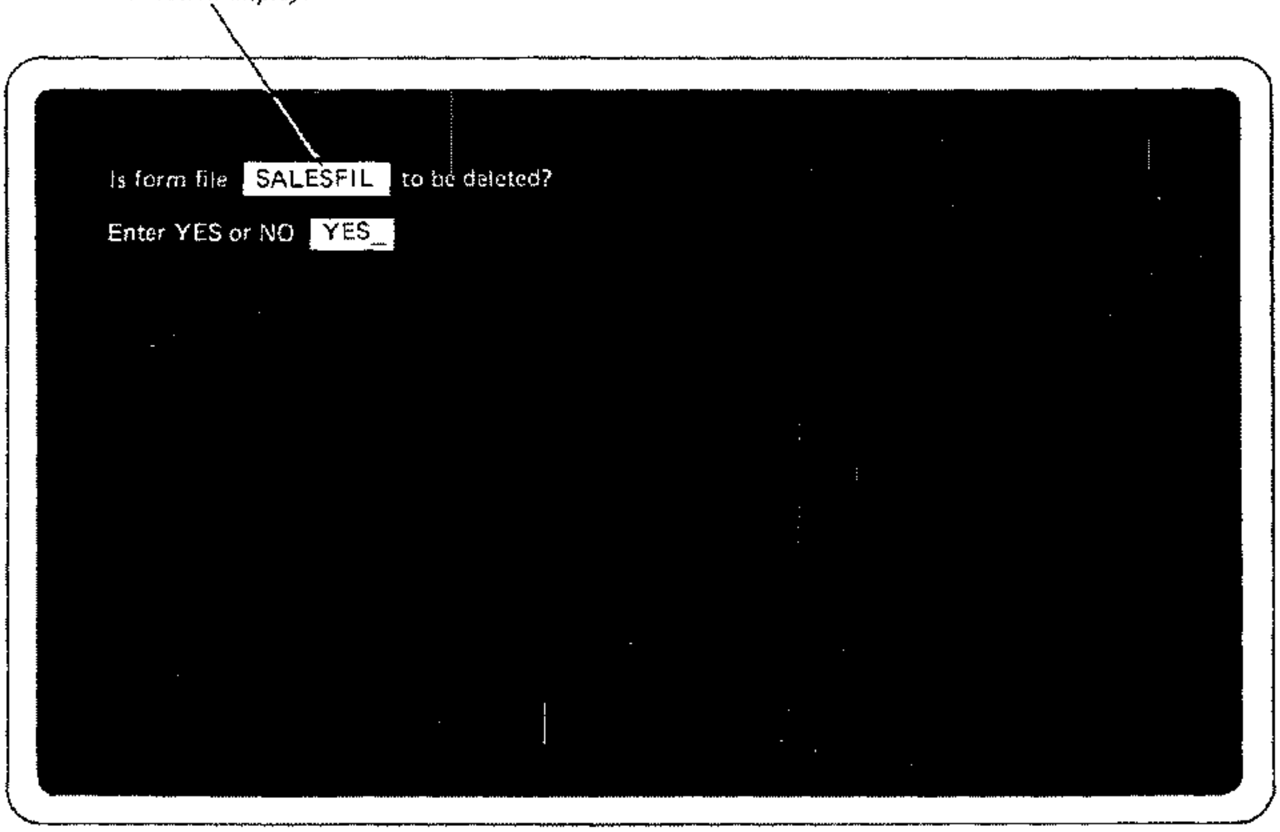
Figure 3-11. Form Listing Output (Continued)



```
Enter the name of the form to be deleted  SALESFORM_
```

Figure 3-12. Form Deletion Display

Name of file specified in function selection display.



```
Is form file  SALESFIL  to be deleted?

Enter YES or NO  YES_
```

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.)

NOTE

> If you use your own procedures as well as the DEL procedures to manipulate the terminal, you must make sure that the communications area (refer to Appendix A) is left in the state that DEL expects and that any escape sequences you transmit to the terminal leave the terminal in a state that DEL expects. The latter point is also true if you use escape sequences with the DEL procedures.

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 a leading plus or minus and 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 following the sign. 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, following the sign.

  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
. . . "OPENFORM" (CHECK STATUS) "OPENTERM" (CHECK STATUS) . . .	. . . "OPENFORM" (CHECK STATUS) "OPENTERM" (CHECK STATUS) . . .
(A) "FINDFORM" (CHECK STATUS) "GETFORM" (CHECK STATUS) "WRITETERM" (CHECK STATUS) (B) "READTERM" (CHECK STATUS)	"SHOWFORM" (CHECK STATUS)
. . .	. . .
"NEXTEDIT" (CHECK STATUS) "NUMRCEDIT" (CHECK STATUS)	"EDITFIELD" (CHECK STATUS)
. . .	. . .
"NEXTEDIT" (CHECK STATUS) "ALPHAEDIT" (CHECK STATUS)	"EDITFIELD" (CHECKSTATUS)
. . . SAME FORM? GO TO (B)* NEW FORM? GO TO (A)	. . .
"CLOSEFORM" (CHECK STATUS) "CLOSETERM" (CHECK STATUS) . . .	"CLOSEFORM" (CHECK STATUS) "CLOSETERM" (CHECK STATUS) . . .

*If it is necessary to clear the screen (clear unprotected fields), it can be done at this point.

Figure 4-1. Procedure Calls in Application Programs

Table 4-1. Summary of Procedure Parameters

PARAMETER	MEANING	DATA ITEM TYPE	AREA/VARIABLE LENGTH
commarea	Name of DEL communication area used for your program. This is a 128-word area whose first entry is *status word* 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
databuffer	Name of program buffer used for input of user data from terminal.	CHARACTER	Length depends upon number of input characters expected plus number of fields plus one.
datalength	Name of variable used to specify length of *databuffer;* DEL also returns length of data actually read to this variable. (If you use for more than one call, you must reset the length since DEL changes it.)	INTEGER	2 bytes (1 word) exactly.
editdef	Name of program buffer to hold specifications for edit procedures.	CHARACTER	72 bytes (characters)
fbufferlength	Name of variable containing length of form buffer; Some DEL procedures also return actual length of form and its allied information to this buffer.	INTEGER	2 bytes (1 word) exactly.
formbuffer	Name of program buffer to which form is moved from form file, and from which form is written to terminal.	CHARACTER	64 bytes minimum.
formfile	Name of area containing MPE formal file designator of current form file.	CHARACTER	35 bytes (characters) maximum. If less than 8 characters, must be terminated by a blank.
formlength	Name of variable to which length of current form is returned.	INTEGER	2 bytes (1 word) exactly.
formname	Name of area containing identity of current form.	CHARACTER	16 bytes (characters) exactly.
nextform	Name of area to which name of next form in a chained series is returned. (You can move to *formname* to request next form in chain.)	CHARACTER	16 bytes (characters) exactly.
statbuffer	Name of buffer to which status of terminal is returned.	CHARACTER	28 bytes minimum.
termfile	Name of area containing MPE formal file designator of terminal file. (Default $STDIN/$STDLIST).	CHARACTER	8 bytes (characters) maximum. If less than 8 characters, must be terminated by a blank.

Note: All parameters must begin on a word boundary.

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 FORMAINT, 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 Intrinsics 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*

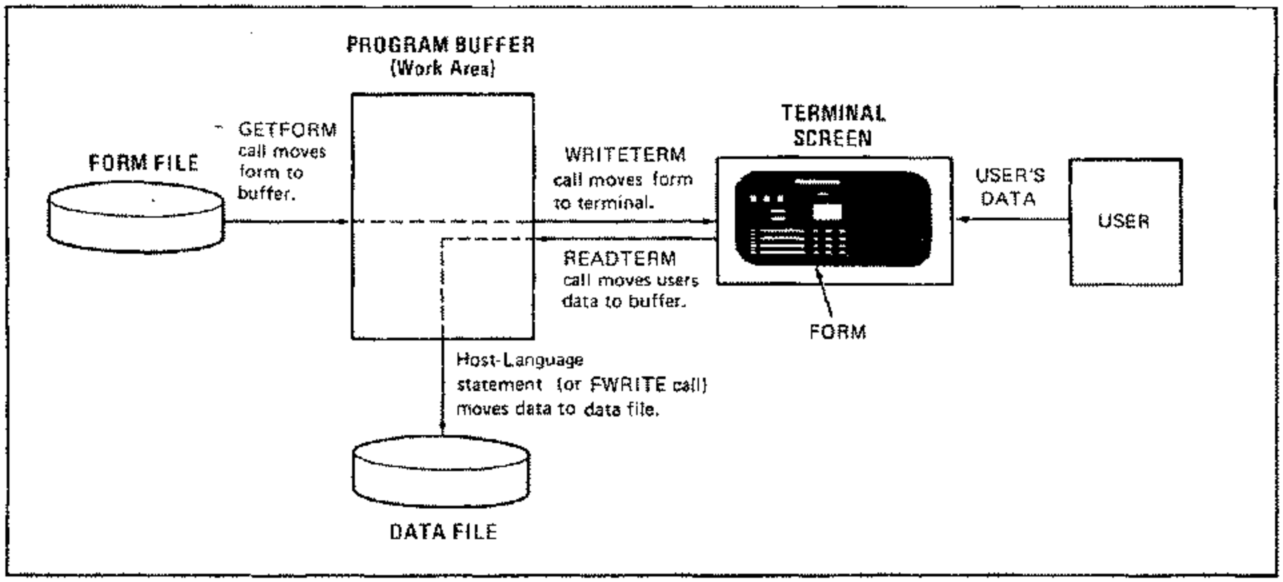


Figure 5-1. Application Program Procedures in Action

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 *commarea formfile.* Example: CALL "COPENFORM" USING MYCOMM MYFORMFL.
FORTRAN	CALL OPENFORM(*commarea,formfile*) Example: CALL OPENFORM(MYCOMM,MYFORMFL)

SPL	OPENFORM(*commarea,formfile*);
	Example: OPENFORM(MYCOMM,MYFORMFL);

BASIC	CALL OPENFORM(*c*(1),*f*$) Example: CALL OPENFORM(C1(1),F$)

The parameters in the OPENFORM calling sequence indicate the following:

PARAMETER	MEANING
commarea *c*	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 *status word* 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.
formfile *f$*	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 informaton 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 *commarea formname formlength nextform.* Example: CALL "CFINDFORM" USING MYCOMM MYFORM LENF NEXTF.
FORTRAN	CALL FINDFORM(*commarea,formname,formlength,nextform*) Example: CALL FINDFORM(MYCOMM,MYFORM,LENF,NEXTF)

SPL	FINDFORM(*commarea,formname,formlength,nextform*);
	Example: FINDFORM(MYCOMM,MYFORM,LENF,NEXTF);

BASIC	CALL FINDFORM(*c*(1),*f1$*,*l*,*f2$*) Example: FINDFORM(C1(1),F1$,L,F2$)

The parameters in this calling sequence indicate:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
formname *f1$*	The name of the area that contains the name of your form. This area is a CHARACTER data item that must be exactly 16 bytes (characters) long.
formlength *l*	The name of a variable to which FINDFORM returns the length of the form accessed. This is a one-word INTEGER data item. This item must be declared usage COMP in COBOL programs.
nextform *f2$*	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. To request the next form, this name must be moved to formname and another call to FINDFORM must be executed.

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. When the complete form has been delivered to the program buffer, word 43 of the communications area (refer to Appendix A) is set to zero. 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. If the program buffer is less than 64 bytes, the status word is set to -4.

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 *commarea formbuffer fbufferlengtb*. Example: CALL "CGETFORM" USING MYCOMM MYFBUF LENB.
FORTRAN	CALL GETFORM(*commarea,formbuffer,fbufferlength*) Example: CALL GETFORM(MYCOMM,MYFBUF,LENB)

SPL	GETFORM(*commarea,formbuffer,fbufferlength*);
	Example: GETFORM(MYCOMM,MYFBUF,LENB);

BASIC	CALL GETFORM(*c*(1),*b*$,*l*) Example: CALL GETFORM(C1(1),B0$,L0)

The parameters in this calling sequence are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
formbuffer *b$*	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.
fbufferlength *l*	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 INTEGER 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 *commarea editdef.* Example: CALL "CNEXTEDIT" USING MYCOMM MYEDEF.
FORTRAN	CALL NEXTEDIT(*commarea,editdef*) Example: CALL NEXTEDIT(MYCOMM,MYEDEF)

SPL	NEXTEDIT(*commarea,editdef*);
	Example: NEXTEDIT(MYCOMM,MYEDEF);

BASIC	CALL NEXTEDIT(*c*(1),*e$*) Example: CALL NEXTEDIT(C1(1),E$)

The parameters in the calling sequence are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion.)
editdef *e$*	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 *commarea.* Example: CALL "CCLOSEFORM" USING MYCOMM.
FORTRAN	CALL CLOSEFORM(*commarea*) Example: CALL CLOSEFORM(MYCOMM)

SPL	CLOSEFORM(*commarea*);
	Example: CLOSEFORM(MYCOMM);

BASIC	CALL CLOSEFORM(*c*(1)) Example: CALL CLOSEFORM(C1(1))

The only parameter in this calling sequence is:

PARAMETER	MEANING
commarea *c*	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 an HP 264x 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 264x, 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 *commarea termfile.* Example: CALL "COPENTERM" USING MYCOMM MYTERMFL.
FORTRAN	CALL OPENTERM(*commarea,termfile*) Example: CALL OPENTERM(MYCOMM,MYTERMFL)

SPL	OPENTERM(*commarea,termfile*);
	Example: OPENTERM(MYCOMM,MYTERMFL);

BASIC	CALL OPENTERM(*c*(1)),*t*$ Example: CALL OPENTERM(C1(1),T$

The parameters for OPENTERM are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
termfile *t*$	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. If less than 8 characters are specified, the name must be terminated with a blank. If the area contains all blanks (no name is specified) the default designator is $STDIN.

NOTE

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

If you want to open a terminal devicefile other than $STDIN/$STDLIST (the session defaults for your terminal), you can use a :FILE command to define some devicefile characteristics. The name passed to OPENTERM in the *termfile* parameter should be used as the formal file designator. The :FILE command DEV= parameter should be used to specify the logical device number of the terminal to be opened; for example, :FILE TERM264X,NEW; DEV=36. If an MPE :DATA command has been entered on the terminal to be opened, the terminal is opened as an old (existing) file and only the device class or logical device number need be defined. Only $STDIN/$STDLIST may be used with the MPE-C operating system.

If the terminal is not $STDIN/$STDLIST, or the :DATA command is not used, and the terminal is to operate at a speed other than 240 characters per second (2400 baud), or at a speed other than its configured default speed, use the Terminal Allocation word in the communications area (refer to Appendix A) to specify the speed. If the configuration value is non-zero and the Terminal Allocation word is zero, the configuration value will override the OPENTERM default of 240 cps.

If the amount of data to be read from any form displayed by the application program is greater than 2000 bytes, you must use a :FILE command REC= parameter to specify the maximum input record size in bytes. For example, if the formal file designator for the terminal is ATERM and the maximum amount of data to be read is 3500 bytes, the file equation should be :FILE ATERM; REC=-3500.

If you want the default values for block mode terminal usage, the communications area should be set to binary zeros before OPENTERM is called. (Refer to Appendix A). If you are using a 2641A, 2645A, 2645K, or 2648A terminal, OPENTERM will set the BLOCK MODE and BLOCK MODE/PAGE strapping programmatically. If you want to suppress this feature, set the Terminal Mode Information word in the communications area to a negative value. If your applications program may be run on an HP 2641, 2645, or 2648 with the MPE-C operating system, be sure that the Terminal Mode Information word is set to a negative value before calling OPENTERM.

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. Only the characters in the buffer are transmitted to the terminal. These include escape sequences to control normal forms display. However, if you want to execute any special escape sequences for terminal control, they must be included in the buffer. You are responsible for insuring that the terminal is in the state that DEL expects it to be when WRITETERM completes execution.

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 *commarea formbuffer fbufferlength.* Example: CALL "CWRITETERM" USING MYCOMM MYFBUF LENB.
FORTRAN	CALL WRITETERM(*commarea,formbuffer,fbufferlength*) Example: CALL WRITETERM(MYCOMM,MYFBUF,LENB)

SPL	WRITETERM(*commarea,formbuffer,fbufferlength*);
	Example: WRITETERM(MYCOMM,MYFBUF,LENB);

BASIC	CALL WRITETERM(*c*(1),*b$*,*l*) Example: CALL WRITETERM(C1(1),B0$,L0)

The parameters in this calling sequence are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
formbuffer *b$*	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.
fbufferlength *l*	The name of a variable that contains the length of the data to be transferred from the buffer. This must be a one-word INTEGER data item. No value is returned.

NOTE

If *commarea* is placed in the DL to DB area by an SPL program, the *formbuffer* program buffer must also be in that area.

DEL prevents messages sent using the MPE :TELL command from being displayed onto forms.

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. If one of the function keys (f1 through f8) is pressed while READTERM is waiting for input, the status word will be set to the negative function key number (-1 through -8).

NOTE

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

During a BLOCK MODE/PAGE read when the G strap is open, the actual transfer of data from a terminal to the computer by READTERM is timed. A data overrun or timeout error occurs if the duration of a read exceeds a computed value. When READTERM detects this condition, it automatically retries the read operation. The number of error recovery retries that READTERM attempts is recorded in words 5, 6, and 7 of the communications area. The default limit to the number of retries is 4, although the transient nature of these errors usually necessitates only one or two retries. If you do not want to use this feature, you can set the Maximum Number of Retries (Word 8 of the communications area) to a negative value.

NOTE

> Before your program can call READTERM, it must have called OPENTERM to open the terminal/file. The buffer length must be at least equal to the number of input data characters plus the number of fields on the screen (to allow for the field separators) plus one (to allow for the record separator).

READING IN PAGE MODE. Note that if you are using READTERM in BLOCK MODE/PAGE, all the fields will be read since READTERM must encounter the record separator at the end of the form to complete the read successfully.

MULTIPOINT. If your application program is running on a multipoint terminal, the maximum number of characters that can be read as data from the unprotected fields of a form is 2048 minus the number of fields minus one.

The calling sequence for READTERM is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CREADTERM" USING *commarea databuffer datalength.* Example: CALL "CREADTERM" USING MYCOMM MYDBUF LENDB.
FORTRAN	CALL READTERM(*commarea,databuffer,datalength*) Example: CALL READTERM(MYCOMM,MYDBUF,LENDB)

SPL	READTERM(*commarea,databuffer,datalength*);
	Example: READTERM(MYCOMM,MYDBUF,LENDB);

BASIC	CALL READTERM(*c*(1),*b*$,*l*) Example: CALL READTERM(C1(1),B1$,L1)

The READTERM parameters are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
databuffer *b$*	The name of the program buffer to receive the user's input from the terminal. The length of this buffer must be at least equal to the number of characters expected in the data entered plus the number of fields plus one. It must be a CHARACTER type data item.
datalength *l*	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 INTEGER type variable. If you are using the same variable for data length in more than one call, you must reset the length since DEL changes it.

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. The first six bytes of the buffer receive the terminal memory size, reported in decimal digits; the remaining bytes receive the Primary Terminal Status flag settings, reported as ASCII 0 (for OFF) and ASCII 1 (for ON). Table 6-1 illustrates how bytes in the terminal status buffer correspond to bits in the Primary Terminal Status bytes described in the terminal owner and programming reference manuals. The HP 2640 (A or B) maintains 18 error flags, while all other terminals maintain 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 *commarea statbuffer.* Example: CALL "CTERMSTATUS" USING MYCOMM MYSTATUS.
FORTRAN	CALL TERMSTATUS(*commarea,statbuffer*) Example: CALL TERMSTATUS(MYCOMM,MYSTATUS)

SPL	TERMSTATUS(*commarea,statbuffer*);
	Example: TERMSTATUS(MYCOMM,MYSTATUS);

BASIC	CALL TERMSTATUS(*c*(1),*s$*) Example: CALL TERMSTATUS(C1(1),S1$)

The TERMSTATUS parameters are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
statbuffer *s$*	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.

Table 6-1. Terminal Status Buffer

	PTS Byte 0 Terminal Memory Size						PTS Byte 1 Keyboard Interface Switches (A-D, Lower Straps)				PTS Byte 2 Keyboard Interface Switches (E-H, Higher Straps)			
PTS Bits							4	3	2	1	4	3	2	1
TSB Bytes	1	2	3	4	5	6	7	8	9	10	11	12	13	14

	PTS Byte 3 Latching Keys			PTS Byte 4 Transfer Pending Flags			PTS Byte 5 Error Flags				PTS Byte 6 Device Transfer Pending Flags			
PTS Bits	3	2	1	3	2	1	4	3	2	1	4	3	2	1
TSB Bytes	14	16	17	18	19	20	21	22	23	24	25	26	27	28

PTS: Primary Terminal Status
TSB: Terminal Status Buffer
Note: PTS Byte 6 does not apply to 2640A and 2640B.

CLOSING A TERMINAL FILE

When your program has finished using the terminal file, it must close 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 *commarea.* Example: CALL "CCLOSETERM" USING MYCOMM.
FORTRAN	CALL CLOSETERM(*commarea*) Example: CALL CLOSETERM(MYCOMM)

SPL	CLOSETERM(*commarea*);
	Example: CLOSETERM(MYCOMM);

BASIC	CALL CLOSETERM(*c*(1)) Example: CALL CLOSETERM(C1(1))

The CLOSETERM parameters are:

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

CLOSETERM returns an HP 2641A, 2645A, 2645K, or 2648A terminal to the state in which it existed before OPENTERM was called. For example, it may reset the terminal to Character Mode. It also returns the $STDIN/$STDLIST devicefile characteristics such as terminal type to their original conditions.

CALLING EDIT PROCEDURES

SECTION VII

DEL provides eight procedures that perform editing functions on data entered at any terminal except an HP 2645K. (Katakana characters cannot be edited with these procedures.) 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) or 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 *commarea editdef databuffer.* Example: CALL "CALPHAEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ALPHAEDIT(*commarea,editdef,databuffer*) Example: CALL ALPHAEDIT(MYCOMM,MYEDEF,MYDBUFF)

SPL	ALPHAEDIT(*commarea,editdef,databuffer*);
	Example: ALPHAEDIT(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL ALPHAEDIT(*c*(1),*e*$,*b*$) Example: CALL ALPHAEDIT(C1(1),ES,B1$)

The parameters in the ALPHAEDIT calling sequence indicate the following:

PARAMETER	MEANING
commarea *c*	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.
editdef *e$*	The name of a CHARACTER-type data item containing the edit specifications as returned by NEXTEDIT. This item must be 72 bytes (characters) long.
databuffer *b$*	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 plus the number of fields plus one.

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 *commarea editdef databuffer.* Example: CALL "CALPHAFILL" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ALPHAFILL(*commarea,editdef,databuffer*) Example: CALL ALPHAFILL(MYCOMM,MYEDEF,MYDBUFF)

SPL	ALPHAFILL(*commarea,editdef,databuffer*);
	Example: ALPHAFILL(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL ALPHAFILL(*c*(1),*e$*,*b$*) Example: CALL ALPHAFILL(C1(1),E$,B1$)

The parameters (*commarea, c*1, *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 *commarea editdef databuffer.* Example: CALL "CANEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ANEDIT(*commarea,editdef,databuffer*) Example: CALL ANEDIT(MYCOMM,MYEDEF,MYDBUFF)

SPL	ANEDIT(*commarea,editdef,databuffer*);
	Example: ANEDIT(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL ANEDIT(*c*(1),*e*$,*b*$) 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 *commarea editdef databuffer.* Example: CALL "CNUMRCEDIT" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL NUMRCEDIT(*commarea,editdef,databuffer*) Example: CALL NUMRCEDIT(MYCOMM,MYEDEF,MYDBUFF)

SPL	NUMRCEDIT(*commarea,editdef,databuffer*);
	Example: NUMRCEDIT(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL NUMRCEDIT(*c*(1),*e*$,*b*$) 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 a leading plus or minus sign, *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 following the sign. The procedure then scans the input field; if any character is not a leading plus or minus sign or 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 other than + or - are allowed.

The calling sequence of the ZEROFILL edit procedure is:

LANGUAGE	CALLING SEQUENCE/EXAMPLE
COBOL	CALL "CZEROFILL" USING *commarea editdef databuffer.* Example: CALL "CZEROFILL" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL ZEROFILL(*commarea,editdef,databuffer*) Example: CALL ZEROFILL(MYCOMM,MYEDEF,MYDBUFF)

SPL	ZEROFILL(*commarea,editdef,databuffer*);
	Example: ZEROFILL(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL ZEROFILL(*c*(1),*e*$,*b*$) 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 *commarea editdef databuffer.* Example: CALL "CNRANGE" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL NRANGE(*commarea,editdef,databuffer*) Example: CALL NRANGE(MYCOMM,MYEDEF,MYDBUFF)

SPL	NRANGE(*commarea,editdef,databuffer*);
	Example: NRANGE(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL NRANGE(*c*(1),*e*$,*b*$) 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 *commarea editdef databuffer.* Example: CALL "CM11CREATE" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL M11CREATE(*commarea,editdef,databuffer*) Example: CALL M11CREATE(MYCOMM,MYEDEF,MYDBUFF)

SPL	M11CREATE(*commarea,editdef,databuffer*);
	Example: M11CREATE(MYCOMM,MYEDEF,MYDBUFF);

BASIC	CALL M11CREATE(*c*(1),*e$*,*b$*) 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 *commarea editdef databuffer.* Example: CALL "CM11VERIFY" USING MYCOMM MYEDEF MYDBUFF.
FORTRAN	CALL M11VERIFY(*commarea,editdef,databuffer*) Example: CALL M11VERIFY(MYCOMM,MYEDEF,MYDBUFF)

SPL	M11VERIFY(*commarea,editdef,databuffer*);
	Example: M11VERIFY(MYCOMM,MYEDEF,MYDBUFF):

BASIC	CALL M11VERIFY(*c*(1),*e$*,*b$*) Example: M11VERIFY(C1(1),E$,B1S)

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 *commarea formname nextform formdatabuffer fdlength.* Example: CALL "CSHOWFORM" USING MYCOMM MYFORM MYFORMN MYFDBUF LENFD.
FORTRAN	CALL SHOWFORM(*commarea,formname,nextform,formdatabuffer,fdlength*) Example: CALL SHOWFORM(MYCOMM,MYFORM,MYFORMN,MYFDBUF,LENFD)

SPL	SHOWFORM(*commarea,formname,nextform,formdatabuffer,fdlength*);
	Example: SHOWFORM(MYCOMM,MYFORM,MYFORMN,MYFDBUF,LENFD);

BASIC	CALL SHOWFORM(*c*(1),*f1$*,*f2$*,*b$*,*l*) Example: CALL SHOWFORM(C1(1),F1$,F2$,B$,L)

The parameters for SHOWFORM are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM procedure discussion in Section V.)
formname *f1$*	The name of the area that contains the identity of the current form. This is a CHARACTER data item exactly 16 bytes long.
nextform *f2$*	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.
formdatabuffer *b$*	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 equal to the number of characters plus the number of fields plus one, a greater length may be required, depending on the size of the screen to be displayed. Must be on word boundary.
fdlength *l*	The name of a variable that contains the length of the program buffer *formdatabuffer*. 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.* After detecting an error in a field, you cannot call EDITFIELD to edit the same field again since it will call NEXTEDIT and advance to the next field.

If the last edit specification for 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 *commarea editdef databuffer.* Example: CALL "CEDITFIELD" USING MYCOMM MYEDEF MYDBUF.
FORTRAN	CALL EDITFIELD(*commarea,editdef,databuffer*) Example: CALL EDITFIELD(MYCOMM,MYEDEF,MYDBUF)

SPL	EDITFIELD(*commarea,editdef,databuffer*);
	Example: EDITFIELD(MYCOMM,MYEDEF,MYDBUF);

BASIC	CALL EDITFIELD(*c*(1),*e*$,*b*$) Example: CALL EDITFIELD(C1(1),E$,B$)

The EDITFIELD parameters are:

PARAMETER	MEANING
commarea *c*	The name of the DEL communication area used for your program. (See OPENFORM discussion in Section V.)
editdef *e*$	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 72 bytes long.
databuffer *b*$	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 plus the number of fields plus one. It must be a CHARACTER data item.

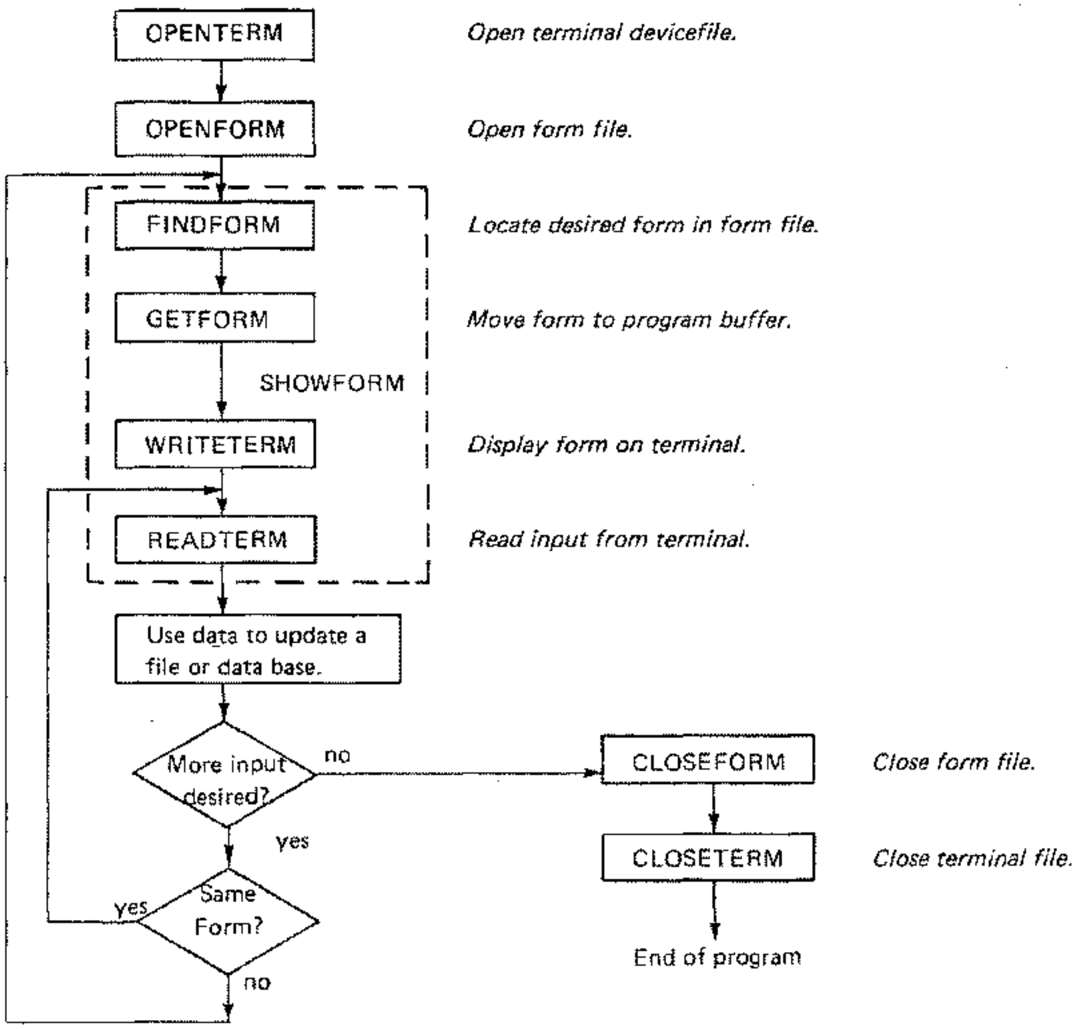
WRITING PROGRAMS

SECTION IX

DEL application programs can be written in COBOL, FORTRAN, BASIC, or SPL to serve a variety of data-entry applications. Examples of calling DEL procedures in the various languages have been provided in previous sections. In this section, the typical flow of a program using DEL procedures is illustrated. An example of a COBOL application program is provided to illustrate some of the programming techniques used for interfacing with DEL.

USING DEL PROCEDURES

The diagram below charts the typical order in which DEL procedures are called by an application program.



The SHOWFORM procedure may be called to perform the functions of FINDFORM, GETFORM, WRITETERM, and READTERM with one procedure call.

If editing procedures are to be executed, the NEXTEDIT procedure and a specific editing procedure may be called after READTERM, or the EDITFIELD procedure may be called.

It is important to remember that all DEL procedure parameters must begin on word boundaries.

COBOL PROGRAM

The following 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 (names 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*

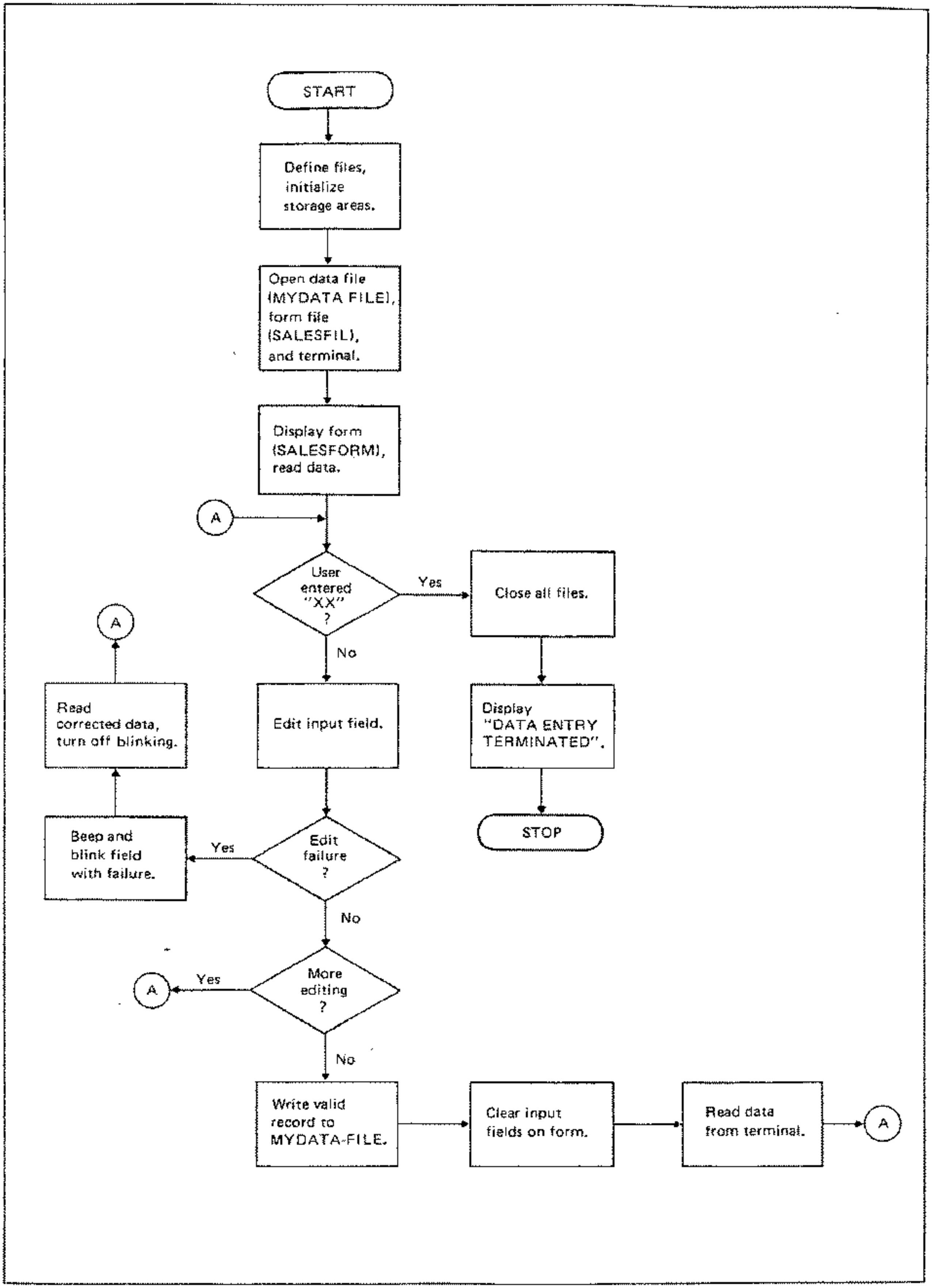
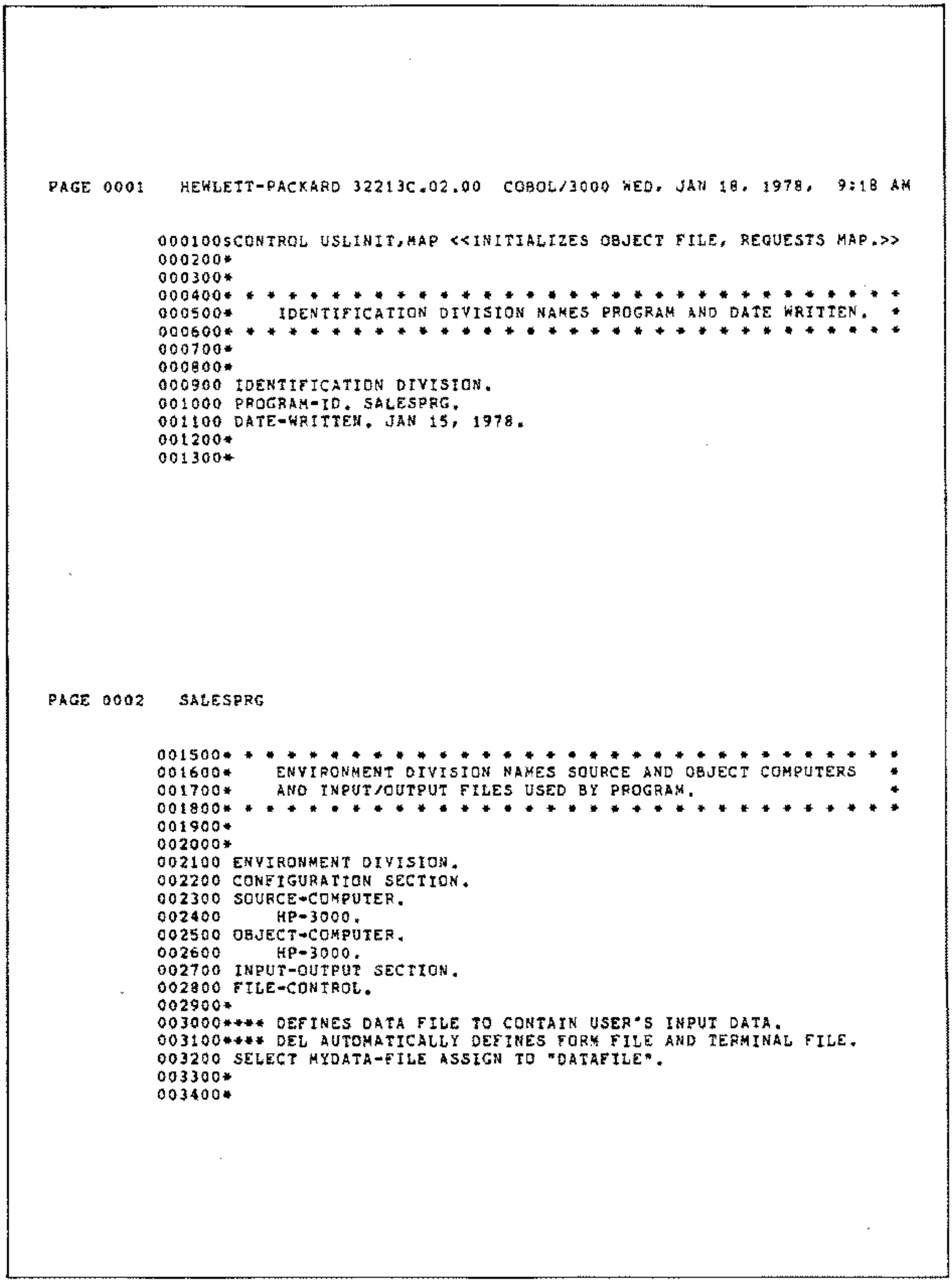

START
Define files, initialize storage areas.
Open data file (MYDATA FILE), form file (SALESFIL), and terminal.
Display form (SALESFORM), read data.
A
User entered "XX" ?
Yes
Close all files.
Display "DATA ENTRY TERMINATED".
STOP
No
Edit input field.
Edit failure ?
Yes
Beep and blink field with failure.
Read corrected data, turn off blinking.
A
No
More editing ?
Yes
A
No
Write valid record to MYDATA-FILE.
Clear input fields on form.
Read data from terminal.
A

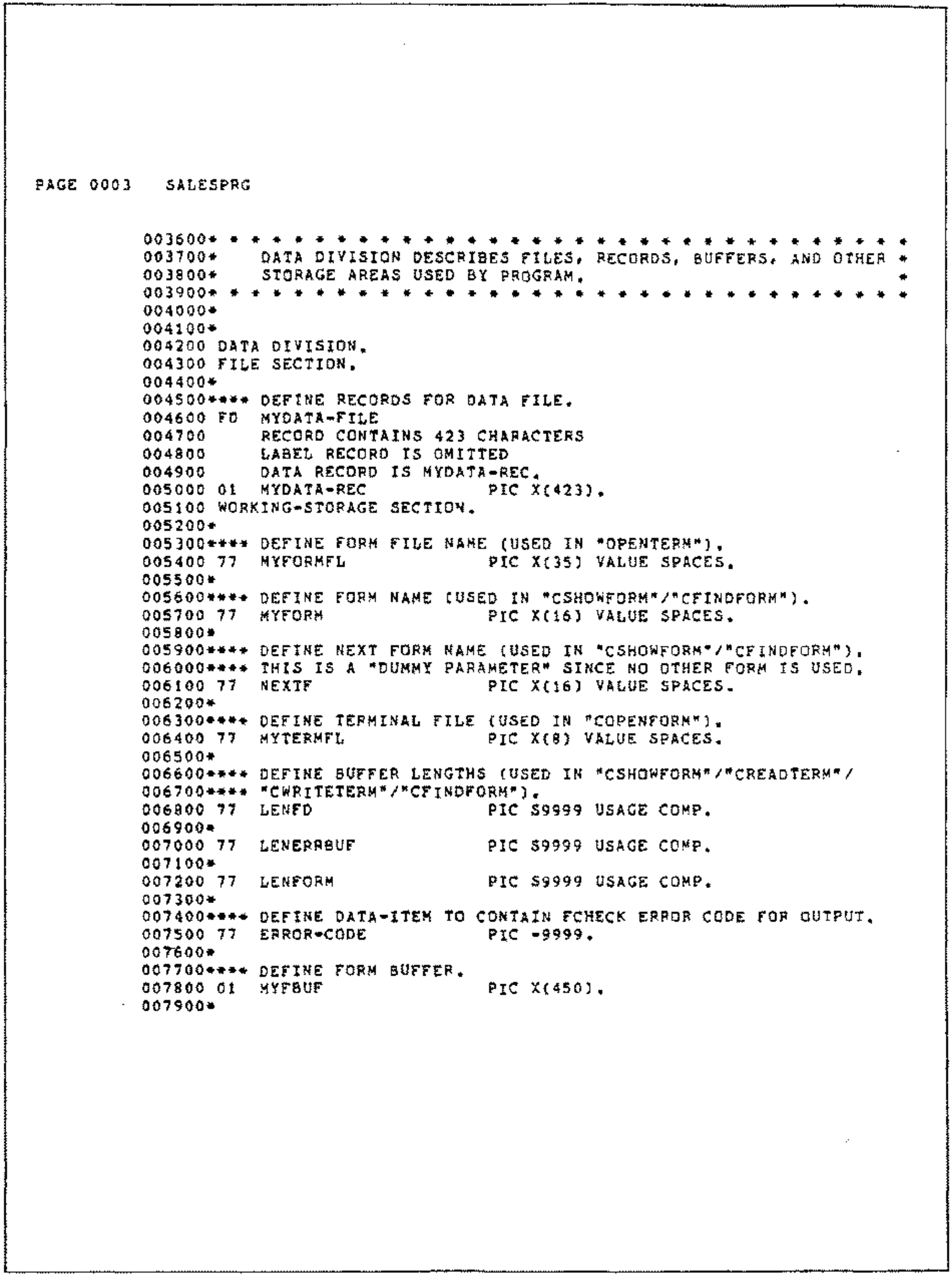
Figure 9-1. COBOL Program Flowchart



```
PAGE 0001   HEWLETT-PACKARD 32213C.02.00  COBOL/3000 WED, JAN 18, 1978,  9:18 AM

          000100$CONTROL USLINIT,MAP <<INITIALIZES OBJECT FILE, REQUESTS MAP.>>
          000200*
          000300*
          000400* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          000500*    IDENTIFICATION DIVISION NAMES PROGRAM AND DATE WRITTEN.  *
          000600* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          000700*
          000800*
          000900 IDENTIFICATION DIVISION.
          001000 PROGRAM-ID. SALESPRG.
          001100 DATE-WRITTEN. JAN 15, 1978.
          001200*
          001300*
```

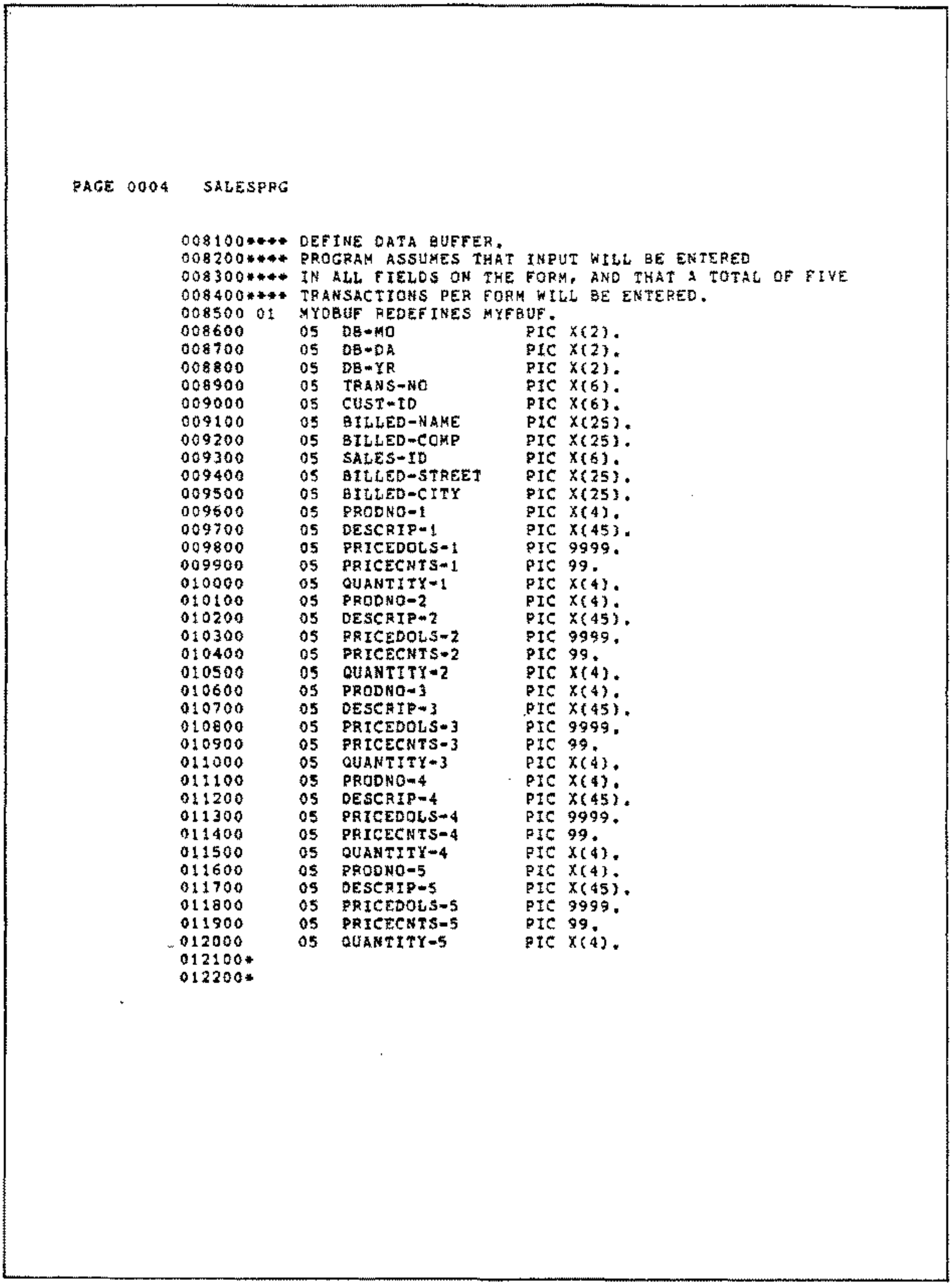
```
PAGE 0002   SALESPRG


          001500* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          001600*    ENVIRONMENT DIVISION NAMES SOURCE AND OBJECT COMPUTERS   *
          001700*    AND INPUT/OUTPUT FILES USED BY PROGRAM.                  *
          001800* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          001900*
          002000*
          002100 ENVIRONMENT DIVISION.
          002200 CONFIGURATION SECTION.
          002300 SOURCE-COMPUTER.
          002400     HP-3000.
          002500 OBJECT-COMPUTER.
          002600     HP-3000.
          002700 INPUT-OUTPUT SECTION.
          002800 FILE-CONTROL.
          002900*
          003000**** DEFINES DATA FILE TO CONTAIN USER'S INPUT DATA.
          003100**** DEL AUTOMATICALLY DEFINES FORM FILE AND TERMINAL FILE.
          003200 SELECT MYDATA-FILE ASSIGN TO "DATAFILE".
          003300*
          003400*
```

Figure 9-2. COBOL Program Listing



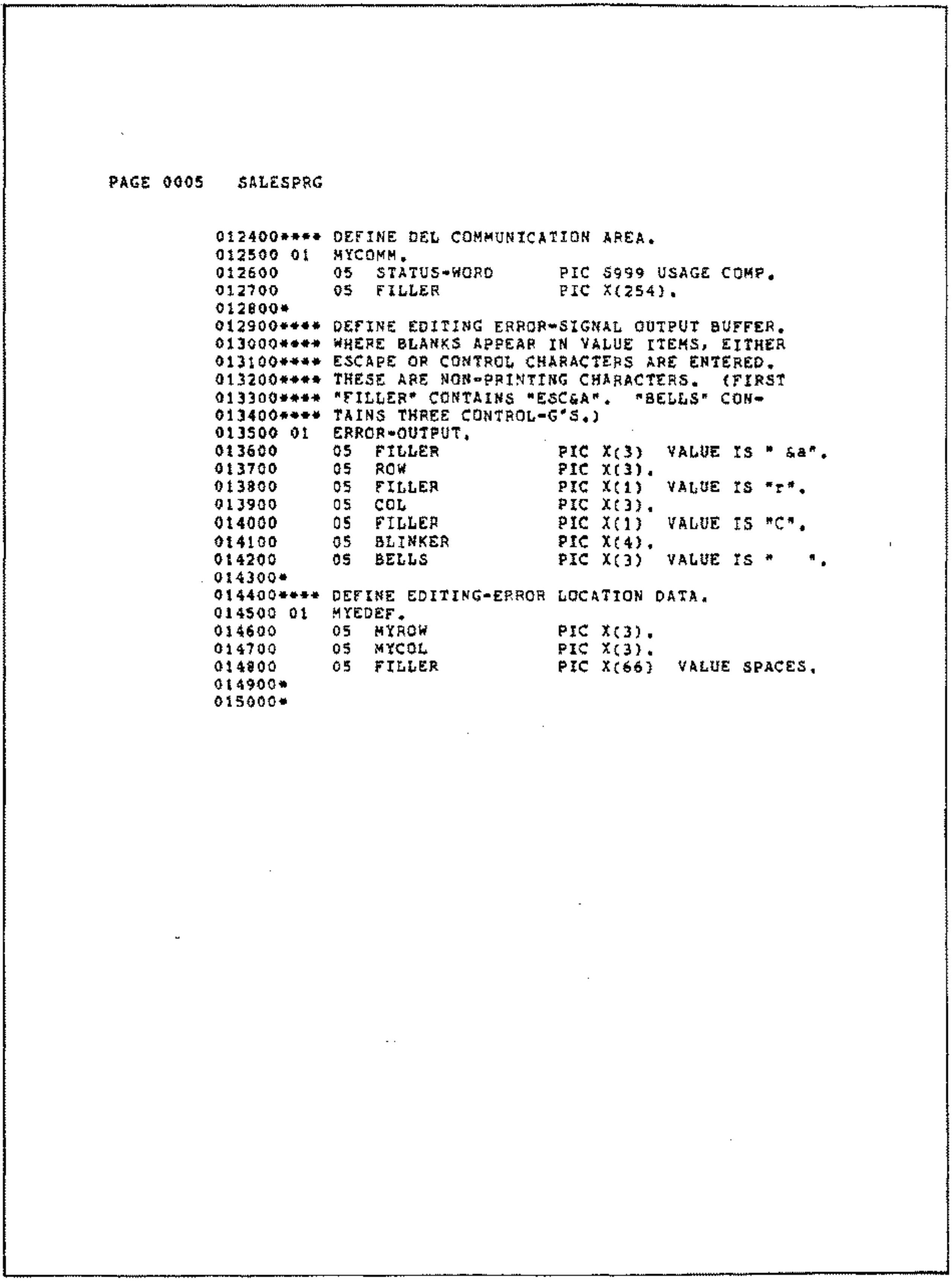
```
PAGE 0003   SALESPRG

         003600* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         003700*    DATA DIVISION DESCRIBES FILES, RECORDS, BUFFERS, AND OTHER *
         003800*    STORAGE AREAS USED BY PROGRAM.                              *
         003900* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         004000*
         004100*
         004200 DATA DIVISION.
         004300 FILE SECTION.
         004400*
         004500**** DEFINE RECORDS FOR DATA FILE.
         004600 FD  MYDATA-FILE
         004700     RECORD CONTAINS 423 CHARACTERS
         004800     LABEL RECORD IS OMITTED
         004900     DATA RECORD IS MYDATA-REC.
         005000 01  MYDATA-REC          PIC X(423).
         005100 WORKING-STORAGE SECTION.
         005200*
         005300**** DEFINE FORM FILE NAME (USED IN "OPENTERM").
         005400 77  MYFORMFL            PIC X(35) VALUE SPACES.
         005500*
         005600**** DEFINE FORM NAME (USED IN "CSHOWFORM"/"CFINDFORM").
         005700 77  MYFORM              PIC X(16) VALUE SPACES.
         005800*
         005900**** DEFINE NEXT FORM NAME (USED IN "CSHOWFORM"/"CFINDFORM").
         006000**** THIS IS A "DUMMY PARAMETER" SINCE NO OTHER FORM IS USED.
         006100 77  NEXTF               PIC X(16) VALUE SPACES.
         006200*
         006300**** DEFINE TERMINAL FILE (USED IN "COPENFORM").
         006400 77  MYTERMFL            PIC X(8) VALUE SPACES.
         006500*
         006600**** DEFINE BUFFER LENGTHS (USED IN "CSHOWFORM"/"CREADTERM"/
         006700**** "CWRITETERM"/"CFINDFORM").
         006800 77  LENFD               PIC S9999 USAGE COMP.
         006900*
         007000 77  LENERRBUF           PIC S9999 USAGE COMP.
         007100*
         007200 77  LENFORM             PIC S9999 USAGE COMP.
         007300*
         007400**** DEFINE DATA-ITEM TO CONTAIN FCHECK ERROR CODE FOR OUTPUT.
         007500 77  ERROR-CODE          PIC -9999.
         007600*
         007700**** DEFINE FORM BUFFER.
         007800 01  MYFBUF              PIC X(450).
         007900*
```

Figure 9-2. COBOL Program Listing (Continued)



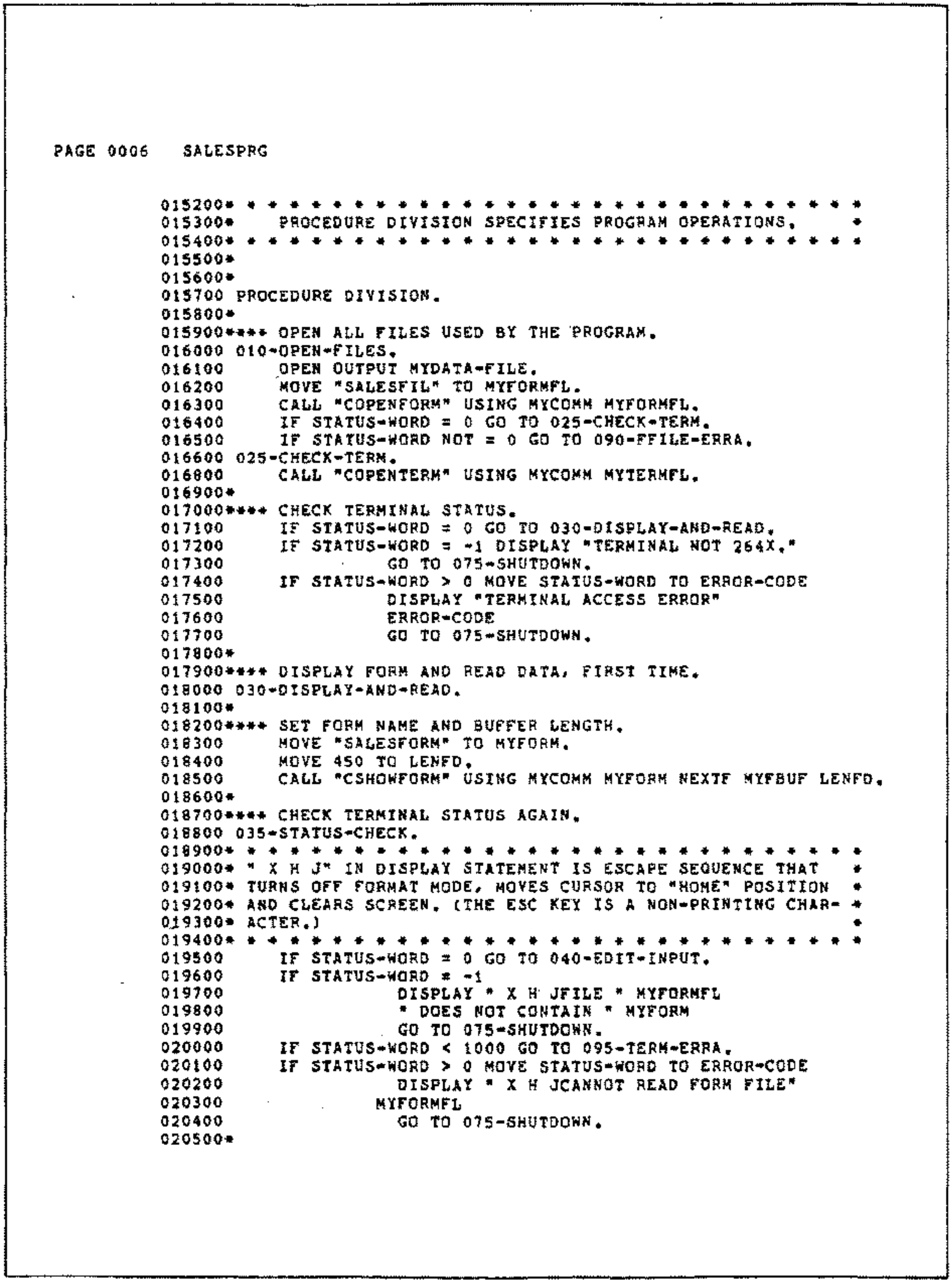
```
PAGE 0004   SALESPRG

          008100**** DEFINE DATA BUFFER.
          008200**** PROGRAM ASSUMES THAT INPUT WILL BE ENTERED
          008300**** IN ALL FIELDS ON THE FORM, AND THAT A TOTAL OF FIVE
          008400**** TRANSACTIONS PER FORM WILL BE ENTERED.
          008500 01  MYDBUF REDEFINES MYFBUF.
          008600     05  DB-MO             PIC X(2).
          008700     05  DB-DA             PIC X(2).
          008800     05  DB-YR             PIC X(2).
          008900     05  TRANS-NO          PIC X(6).
          009000     05  CUST-ID           PIC X(6).
          009100     05  BILLED-NAME       PIC X(25).
          009200     05  BILLED-COMP       PIC X(25).
          009300     05  SALES-ID          PIC X(6).
          009400     05  BILLED-STREET     PIC X(25).
          009500     05  BILLED-CITY       PIC X(25).
          009600     05  PRODNO-1          PIC X(4).
          009700     05  DESCRIP-1         PIC X(45).
          009800     05  PRICEDOLS-1       PIC 9999.
          009900     05  PRICECNTS-1       PIC 99.
          010000     05  QUANTITY-1        PIC X(4).
          010100     05  PRODNO-2          PIC X(4).
          010200     05  DESCRIP-2         PIC X(45).
          010300     05  PRICEDOLS-2       PIC 9999.
          010400     05  PRICECNTS-2       PIC 99.
          010500     05  QUANTITY-2        PIC X(4).
          010600     05  PRODNO-3          PIC X(4).
          010700     05  DESCRIP-3         PIC X(45).
          010800     05  PRICEDOLS-3       PIC 9999.
          010900     05  PRICECNTS-3       PIC 99.
          011000     05  QUANTITY-3        PIC X(4).
          011100     05  PRODNO-4          PIC X(4).
          011200     05  DESCRIP-4         PIC X(45).
          011300     05  PRICEDOLS-4       PIC 9999.
          011400     05  PRICECNTS-4       PIC 99.
          011500     05  QUANTITY-4        PIC X(4).
          011600     05  PRODNO-5          PIC X(4).
          011700     05  DESCRIP-5         PIC X(45).
          011800     05  PRICEDOLS-5       PIC 9999.
          011900     05  PRICECNTS-5       PIC 99.
          012000     05  QUANTITY-5        PIC X(4).
          012100*
          012200*
```

Figure 9-2. COBOL Program Listing (Continued)



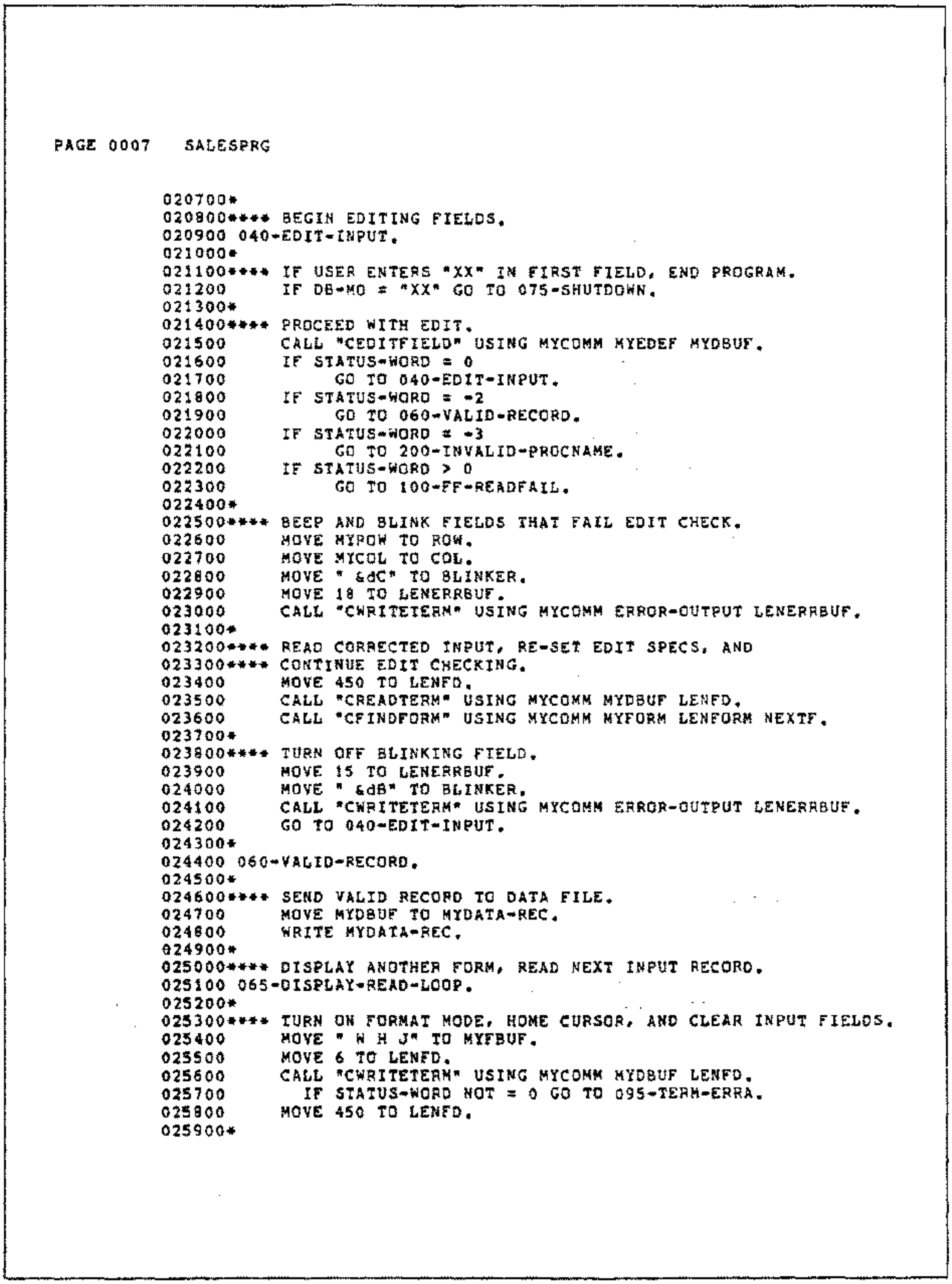
```
PAGE 0005   SALESPRG

          012400**** DEFINE DEL COMMUNICATION AREA.
          012500 01  MYCOMM.
          012600         05  STATUS-WORD        PIC S999 USAGE COMP.
          012700         05  FILLER             PIC X(254).
          012800*
          012900**** DEFINE EDITING ERROR-SIGNAL OUTPUT BUFFER.
          013000**** WHERE BLANKS APPEAR IN VALUE ITEMS, EITHER
          013100**** ESCAPE OR CONTROL CHARACTERS ARE ENTERED.
          013200**** THESE ARE NON-PRINTING CHARACTERS.  (FIRST
          013300**** "FILLER" CONTAINS "ESC&A".  "BELLS" CON-
          013400**** TAINS THREE CONTROL-G'S.)
          013500 01  ERROR-OUTPUT.
          013600         05  FILLER             PIC X(3)  VALUE IS " &a".
          013700         05  ROW                PIC X(3).
          013800         05  FILLER             PIC X(1)  VALUE IS "r".
          013900         05  COL                PIC X(3).
          014000         05  FILLER             PIC X(1)  VALUE IS "C".
          014100         05  BLINKER            PIC X(4).
          014200         05  BELLS              PIC X(3)  VALUE IS "   ".
          014300*
          014400**** DEFINE EDITING-ERROR LOCATION DATA.
          014500 01  MYEDEF.
          014600         05  MYROW              PIC X(3).
          014700         05  MYCOL              PIC X(3).
          014800         05  FILLER             PIC X(66)  VALUE SPACES.
          014900*
          015000*
```

Figure 9-2. COBOL Program Listing (Continued)



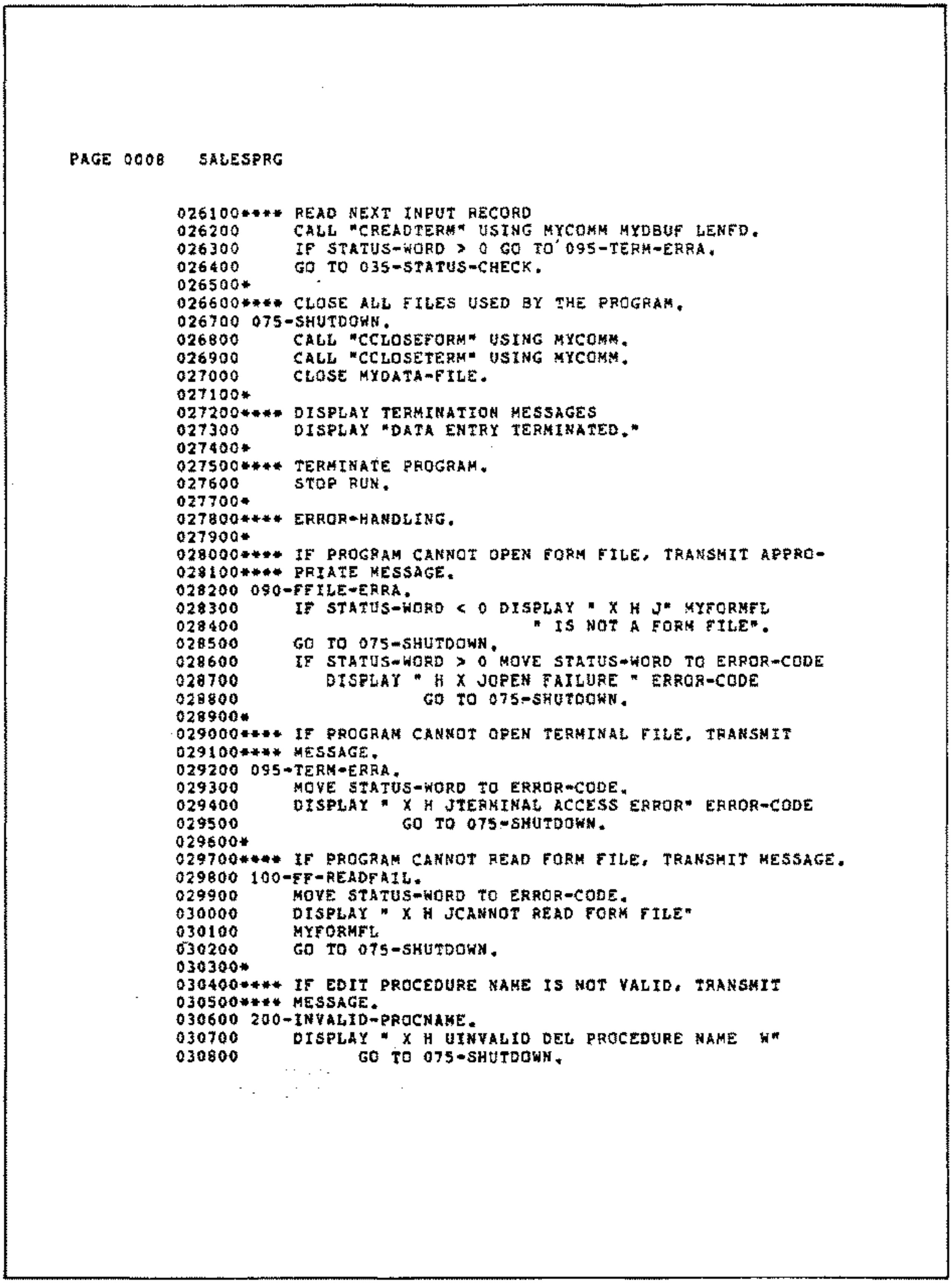
```
PAGE 0006   SALESPRG

        015200* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        015300*    PROCEDURE DIVISION SPECIFIES PROGRAM OPERATIONS,      *
        015400* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        015500*
        015600*
        015700 PROCEDURE DIVISION.
        015800*
        015900**** OPEN ALL FILES USED BY THE PROGRAM.
        016000 010-OPEN-FILES.
        016100     OPEN OUTPUT MYDATA-FILE.
        016200     MOVE "SALESFIL" TO MYFORMFL.
        016300     CALL "COPENFORM" USING MYCOMM MYFORMFL.
        016400     IF STATUS-WORD = 0 GO TO 025-CHECK-TERM.
        016500     IF STATUS-WORD NOT = 0 GO TO 090-FFILE-ERRA.
        016600 025-CHECK-TERM.
        016800     CALL "COPENTERM" USING MYCOMM MYTERMFL.
        016900*
        017000**** CHECK TERMINAL STATUS.
        017100     IF STATUS-WORD = 0 GO TO 030-DISPLAY-AND-READ.
        017200     IF STATUS-WORD = -1 DISPLAY "TERMINAL NOT 264X."
        017300               GO TO 075-SHUTDOWN.
        017400     IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
        017500               DISPLAY "TERMINAL ACCESS ERROR"
        017600               ERROR-CODE
        017700               GO TO 075-SHUTDOWN.
        017800*
        017900**** DISPLAY FORM AND READ DATA, FIRST TIME.
        018000 030-DISPLAY-AND-READ.
        018100*
        018200**** SET FORM NAME AND BUFFER LENGTH.
        018300     MOVE "SALESFORM" TO MYFORM.
        018400     MOVE 450 TO LENFD.
        018500     CALL "CSHOWFORM" USING MYCOMM MYFORM NEXTF MYFBUF LENFD.
        018600*
        018700**** CHECK TERMINAL STATUS AGAIN.
        018800 035-STATUS-CHECK.
        018900* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        019000* " X H J" IN DISPLAY STATEMENT IS ESCAPE SEQUENCE THAT    *
        019100* TURNS OFF FORMAT MODE, MOVES CURSOR TO "HOME" POSITION   *
        019200* AND CLEARS SCREEN. (THE ESC KEY IS A NON-PRINTING CHAR- *
        019300* ACTER.)                                                  *
        019400* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        019500     IF STATUS-WORD = 0 GO TO 040-EDIT-INPUT.
        019600     IF STATUS-WORD = -1
        019700                DISPLAY " X H JFILE " MYFORMFL
        019800                " DOES NOT CONTAIN " MYFORM
        019900                GO TO 075-SHUTDOWN.
        020000     IF STATUS-WORD < 1000 GO TO 095-TERM-ERRA.
        020100     IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
        020200                DISPLAY " X H JCANNOT READ FORM FILE"
        020300              MYFORMFL
        020400                GO TO 075-SHUTDOWN.
        020500*
```

Figure 9-2. COBOL Program Listing (Continued)



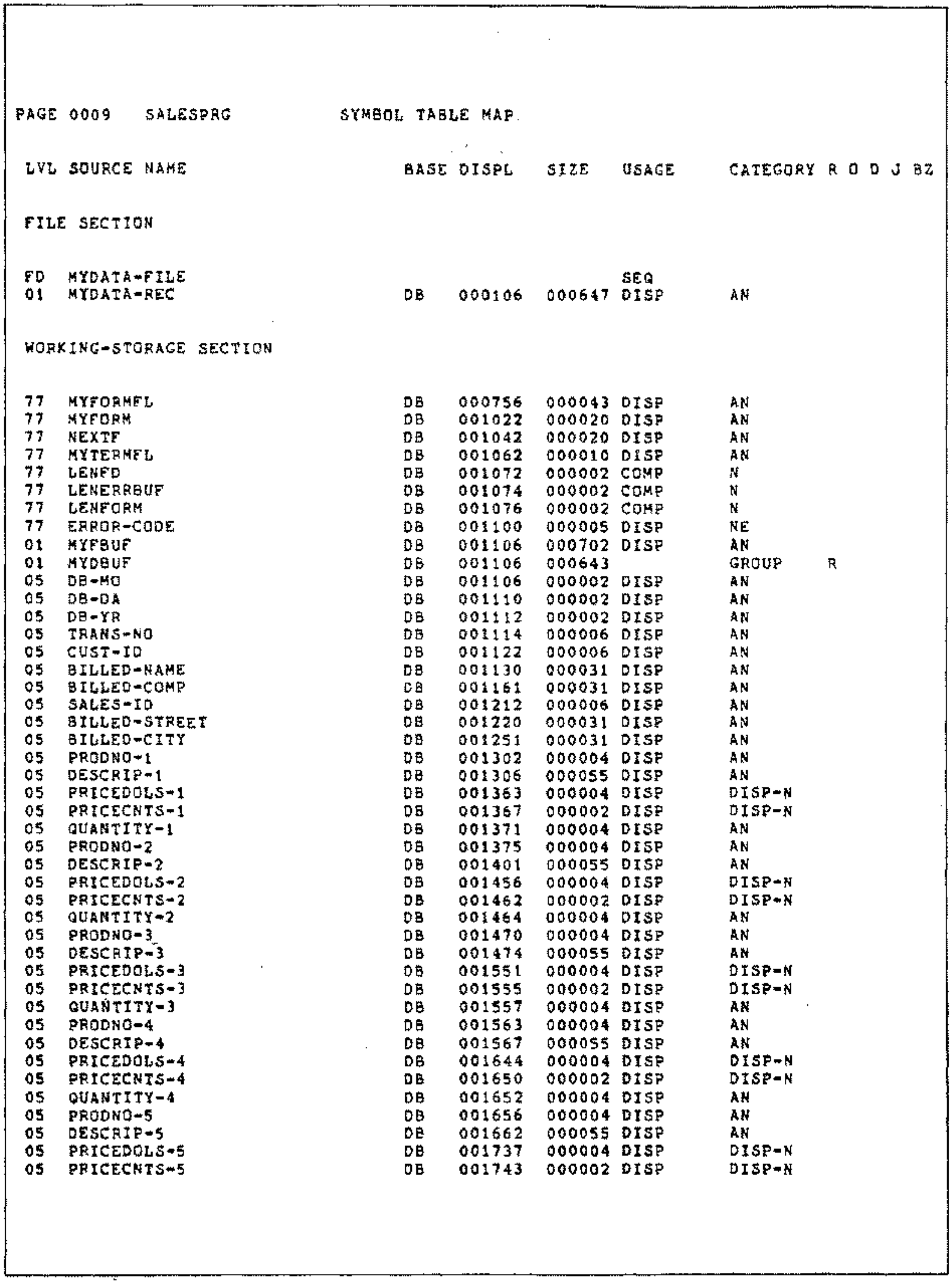
```
PAGE 0007   SALESPRG

          020700*
          020800**** BEGIN EDITING FIELDS.
          020900 040-EDIT-INPUT.
          021000*
          021100**** IF USER ENTERS "XX" IN FIRST FIELD, END PROGRAM.
          021200      IF DB-MO = "XX" GO TO 075-SHUTDOWN.
          021300*
          021400**** PROCEED WITH EDIT.
          021500      CALL "CEDITFIELD" USING MYCOMM MYEDEF MYDBUF.
          021600      IF STATUS-WORD = 0
          021700           GO TO 040-EDIT-INPUT.
          021800      IF STATUS-WORD = -2
          021900           GO TO 060-VALID-RECORD.
          022000      IF STATUS-WORD = -3
          022100           GO TO 200-INVALID-PROCNAME.
          022200      IF STATUS-WORD > 0
          022300           GO TO 100-FF-READFAIL.
          022400*
          022500**** BEEP AND BLINK FIELDS THAT FAIL EDIT CHECK.
          022600      MOVE MYROW TO ROW.
          022700      MOVE MYCOL TO COL.
          022800      MOVE " &dC" TO BLINKER.
          022900      MOVE 18 TO LENERRBUF.
          023000      CALL "CWRITETERM" USING MYCOMM ERROR-OUTPUT LENERRBUF.
          023100*
          023200**** READ CORRECTED INPUT, RE-SET EDIT SPECS, AND
          023300**** CONTINUE EDIT CHECKING.
          023400      MOVE 450 TO LENFD.
          023500      CALL "CREADTERM" USING MYCOMM MYDBUF LENFD.
          023600      CALL "CFINDFORM" USING MYCOMM MYFORM LENFORM NEXTF.
          023700*
          023800**** TURN OFF BLINKING FIELD.
          023900      MOVE 15 TO LENERRBUF.
          024000      MOVE " &dB" TO BLINKER.
          024100      CALL "CWRITETERM" USING MYCOMM ERROR-OUTPUT LENERRBUF.
          024200      GO TO 040-EDIT-INPUT.
          024300*
          024400 060-VALID-RECORD.
          024500*
          024600**** SEND VALID RECORD TO DATA FILE.
          024700      MOVE MYDBUF TO MYDATA-REC.
          024800      WRITE MYDATA-REC.
          024900*
          025000**** DISPLAY ANOTHER FORM, READ NEXT INPUT RECORD.
          025100 065-DISPLAY-READ-LOOP.
          025200*
          025300**** TURN ON FORMAT MODE, HOME CURSOR, AND CLEAR INPUT FIELDS.
          025400      MOVE " W H J" TO MYFBUF.
          025500      MOVE 6 TO LENFD.
          025600      CALL "CWRITETERM" USING MYCOMM MYDBUF LENFD.
          025700        IF STATUS-WORD NOT = 0 GO TO 095-TERM-ERRA.
          025800      MOVE 450 TO LENFD.
          025900*
```

Figure 9-2. COBOL Program Listing (Continued)



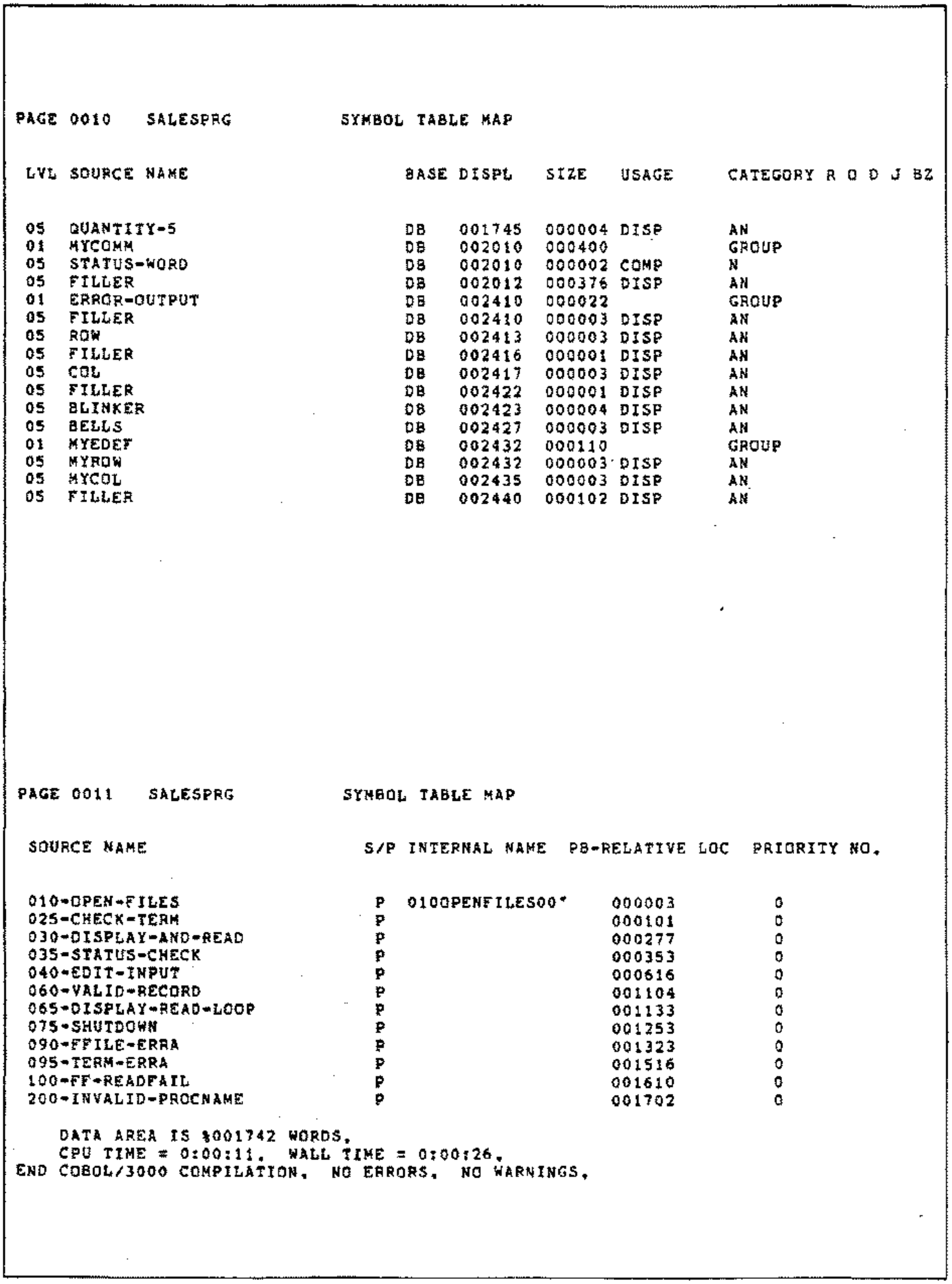
```
PAGE 0008   SALESPRG

          026100**** READ NEXT INPUT RECORD
          026200     CALL "CREADTERM" USING MYCOMM MYDBUF LENFD.
          026300     IF STATUS-WORD > 0 GO TO 095-TERM-ERRA.
          026400     GO TO 035-STATUS-CHECK.
          026500*
          026600**** CLOSE ALL FILES USED BY THE PROGRAM.
          026700 075-SHUTDOWN.
          026800     CALL "CCLOSEFORM" USING MYCOMM.
          026900     CALL "CCLOSETERM" USING MYCOMM.
          027000     CLOSE MYDATA-FILE.
          027100*
          027200**** DISPLAY TERMINATION MESSAGES
          027300     DISPLAY "DATA ENTRY TERMINATED."
          027400*
          027500**** TERMINATE PROGRAM.
          027600     STOP RUN.
          027700*
          027800**** ERROR-HANDLING.
          027900*
          028000**** IF PROGRAM CANNOT OPEN FORM FILE, TRANSMIT APPRO-
          028100**** PRIATE MESSAGE.
          028200 090-FFILE-ERRA.
          028300     IF STATUS-WORD < 0 DISPLAY " X H J" MYFORMFL
          028400                                " IS NOT A FORM FILE".
          028500     GO TO 075-SHUTDOWN.
          028600     IF STATUS-WORD > 0 MOVE STATUS-WORD TO ERROR-CODE
          028700        DISPLAY " H X JOPEN FAILURE " ERROR-CODE
          028800                 GO TO 075-SHUTDOWN.
          028900*
          029000**** IF PROGRAM CANNOT OPEN TERMINAL FILE, TRANSMIT
          029100**** MESSAGE.
          029200 095-TERM-ERRA.
          029300     MOVE STATUS-WORD TO ERROR-CODE.
          029400     DISPLAY " X H JTERMINAL ACCESS ERROR" ERROR-CODE
          029500               GO TO 075-SHUTDOWN.
          029600*
          029700**** IF PROGRAM CANNOT READ FORM FILE, TRANSMIT MESSAGE.
          029800 100-FF-READFAIL.
          029900     MOVE STATUS-WORD TO ERROR-CODE.
          030000     DISPLAY " X H JCANNOT READ FORM FILE"
          030100     MYFORMFL
          030200     GO TO 075-SHUTDOWN.
          030300*
          030400**** IF EDIT PROCEDURE NAME IS NOT VALID, TRANSMIT
          030500**** MESSAGE.
          030600 200-INVALID-PROCNAME.
          030700     DISPLAY " X H UINVALID DEL PROCEDURE NAME  W"
          030800           GO TO 075-SHUTDOWN.
```

Figure 9-2. COBOL Program Listing (Continued)



```
PAGE 0009   SALESPRG          SYMBOL TABLE MAP

 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  MYDBUF                         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



```
PAGE 0010   SALESPRG           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
```

```
PAGE 0011   SALESPRG           SYMBOL TABLE MAP

 SOURCE NAME                       S/P INTERNAL NAME  PB-RELATIVE LOC  PRIORITY NO.

 010-OPEN-FILES                     P  010OPENFILES00'     000003           0
 025-CHECK-TERM                     P                      000101           0
 030-DISPLAY-AND-READ               P                      000277           0
 035-STATUS-CHECK                   P                      000353           0
 040-EDIT-INPUT                     P                      000616           0
 060-VALID-RECORD                   P                      001104           0
 065-DISPLAY-READ-LOOP              P                      001133           0
 075-SHUTDOWN                       P                      001253           0
 090-FFILE-ERRA                     P                      001323           0
 095-TERM-ERRA                      P                      001516           0
 100-FF-READFAIL                    P                      001610           0
 200-INVALID-PROCNAME               P                      001702           0

    DATA AREA IS %001742 WORDS.
    CPU TIME = 0:00:11.  WALL TIME = 0:00:26.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.
```

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

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
```

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 FORMAINT (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.

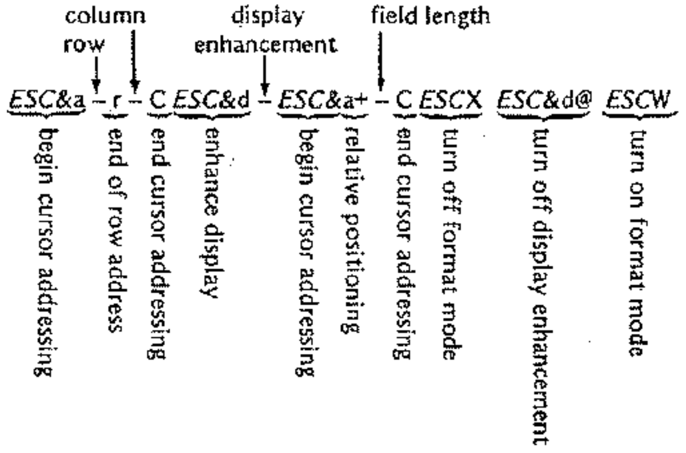
In summary, the table look-up manipulation is up to your application program and should be handled in the best manner suitable for that program.

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:



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	*Esc*c
Turn off Format Mode	*Esc*X
Cursor Addressing	*Esc*&a23rOC
Error Message	" "
Turn on Format Mode	*Esc*W
Unlock Keyboard	*Esc*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	*Esc*c
Turn off Format Mode	*Esc*X
Cursor Down	*Esc*B
Insert Line	*Esc*L (positioned at beginning of line.)
Error Message	" "
Turn on Format Mode	*Esc*W
Unlock Keyboard	*Esc*b

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 FORMAINT. If an error is caused by incorrect data that you entered, the field containing the data blinks (if your terminal has the blinking feature). FORMAINT 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 FORMAINT function:

1. INPUT DEVICE IS NOT A 264X

   FORMAINT checks the input device, which must be an HP 264x 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 FORMAINT cannot open your terminal as an input file, the above message appears on the screen and FORMAINT terminates. The *nnnn* parameter indicates an error code returned to FORMAINT by the MPE intrinsic FCHECK. Refer to *MPE Intrinsics 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 Intrinsics Reference Manual* for meanings of error codes listed under the FCHECK intrinsic. If you enter a YES in response to the prompt ABORT?, FORMAINT 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, FORMAINT attempts to continue the current operation, ignoring the terminal error.

Table 10-1. Error Messages Generated by FORMAINT

MESSAGE	DISPLAY FIG #	MEANING	CORRECTIVE ACTION
BOTH D AND G STRAPS MUST BE CLOSED	—	In order to operate in line mode, FORMAINT requires that the D and G straps be closed on 2640A, 2640B and 2644A terminals.	Put straps in and rerun FORMAINT.

Table 10-1. Error Messages Generated by FORMAINT (continued)

MESSAGE	DISPLAY FIG. #	MEANING	CORRECTIVE ACTION
EDIT PROCEDURE NAME IS INVALID	3-7	You entered an edit procedure name that does not begin with an alphabetic character, or that contains a special character; or you omitted a procedure name where one is required.	Refer to *Edit Procedure Name* in Section III and enter valid edit procedure name.
FILE IS INACCESSIBLE	3-4	Another user is presently accessing the form file. FORMAINT opens the form file for exclusive access by one user at a time. NOTE: Many users, however, may access the form file for data entry through application programs.	When file becomes accessible, enter the valid name of your form file in the designated input field and press ENTER.
FILE CANNOT BE OPENED ERROR CODE=*nnnn*	3-4	FORMAINT cannot open the form file you specified. *nnnn* parameter indicates an error code returned by the FCHECK intrinsic.	Refer to Appendix G for error code meanings listed under FCHECK intrinsic. Correct error condition, then enter the valid name of your form file in the designated input field and press ENTER.
FILE IS NOT A FORMS FILE	3-4	You specified an existing file that is not a form file. FORMAINT operates on form files only so that data in another file is not destroyed by accident.	The file may be another program/data file on the system. Check accuracy of file name, enter the valid name of your form file in the designated input field, and press ENTER.
FILE NAME REQUIRED	3-4	You did not enter a form file name.	Enter the valid name of your form file in the designated input field and press ENTER.
FLAG # IS INVALID	3-7	You entered a number other than 1 through 16 or a blank in response to a prompt for a flag.	Refer to *Test Flag # Before Edit* in Section III. If a flag is required, enter a number from 1 through 16. Otherwise, enter a blank.
FORM ALREADY EXISTS IN *filename*	3-5	You specified a form that already exists in the form file.	Check accuracy of form name and enter valid form name in the designated input field. To select another function, press *f*8^c (on 2640) or *f*8. Function Selection Display will appear on terminal.

Table 10-1. Error Messages Generated by FORMAINT (continued)

MESSAGE	DISPLAY FIG. #	MEANING	CORRECTIVE ACTION
FORM DEFINITION EXCEEDS SYSTEM CAPABILITY	3-7	Form contains one or more lines that exceed 216 characters in length. (Limitation on MPE-C operating system.)	Decrease the size of the line and reenter the form.
FORM DOES NOT EXIST IN *filename*	3-9 3-10 3-12	You specified the name of a form that does not reside in the form file.	Check accuracy of form name and enter valid name of form to be modified, displayed, or deleted.
FORM FILE ACCESS ERROR CODE = *error number*	3-4	FORMAINT encountered a read or write error when accessing your form file.	Refer to Appendix G for *error number* meaning listed under FCHECK intrinsic. Correct error condition and run FORMAINT again.
FORM FILE IS FULL	3-5	There is no space available in the specified form file for a new form.	Specify another form file to contain the new form by entering the valid form file name in the designated input field and pressing ENTER.
FORM FILE NAME INVALID	3-4	You entered an invalid form file name.	Refer to Section III for form file name requirements and enter the valid name of your form file in the designated input field; then press ENTER.
FORMAINT MUST HAVE UPDATE ACCESS TO FORMS FILE	3-4	Form file does not allow read/write access.	Change the access mode of the form file to INOUT (read/write).
INVALID ROW ADDRESS	3-7 3-10	Data in form file may be invalid.	If output during DISPLAY function, delete and reenter form. If output during Edit specification, check form by displaying and delete and reenter if necessary.
FORM NAME REQUIRED	3-5	You did not enter a form name.	Refer to *Form Name* in Section III and enter valid form name in the designated input field.
LIST FILE CANNOT BE ACCESSED, ERROR CODE=*nnnn*	3-10	The output file for the form listing cannot be accessed. *nnnn* is the error code returned to FORMAINT by the FCHECK intrinsic.	Refer to *MPE Intrinsic Reference Manual* for error code meanings listed under FCHECK intrinsic. Correct error condition; then enter valid file name for output file.
NO FUNCTION SELECTED	3-4	You did not select a function from the list displayed.	Enter an X in front of the desired function, and press ENTER.

Table 10-1. Error Messages Generated by FORMAINT (continued)

MESSAGE	DISPLAY FIG #	MEANING	CORRECTIVE ACTION
NO ROOM IN FILE FOR MODIFY	3-9	Form file is too small to accommodate newly modified form.	Specify another form file to contain the new version of the form and recreate the form.
SELECT ONE – RANGE CHECK, FILE LOOK-UP OR EDIT DEFINED DATA	3-7	You specified *more than one* of the following entries for an input field: 1) Range Check 2) Look-up Procedure File Name 3) Characters (for edit-defined data)	Select only one of the entries and enter the appropriate information.

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.

If you want to operate in LINE mode, you should set word 3 of the communications area to a negative value before OPENTERM is executed.

If you want the default values for block mode operation, you should set the first ten words of the communications area to binary zeros.

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	Binary	Terminal allocation information
5	Binary	Data overrun logging
6	Binary	Read timeout logging
7	Binary	Other data error logging
8	Binary	Maximum number of retries
9	Binary	Suppress messages in OPENTERM and CLOSETERM and enable autoread feature in READTERM
10	Binary	Environment information
11	Binary	Reserved
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 + Number of Fields
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 When zero, all of form has been displayed.
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

CONTENTS OF 3 THROUGH 10

A detailed description of the way in which DEL uses words 3 through 10 of the communications area is provided below. You may want to modify certain bits in these words if you do not want the default terminal modes that DEL provides.

C657
62V

TERMINAL MODE INFORMATION (WORD 3)

bits:

0	1	2	3	4 5 6 7 8 9	10 11 12	13	14	15
P	E	F	x	TERMTYPE from logon	0=2640A 1=2640B 2=2644A 3=2645A 3=2645K 3-2641A 3=2648	B	G	D

P: 0=allow 2641, 2645, 2648 programmatic switch settings (positive);
1=use 264x physical switch settings (negative)

E: 0=echo was on before OPENTERM, now off;
1=echo was off

F: 0="DEV=. . ." used in file equate, not $STDIN;
1=file is $STDIN/$STDLIST

B: 1=Block Mode key down at last status request in OPENTERM;
0=Block Mode key up.

G: 1=strap G out (required for block mode/page)
0=strap G in (required for block mode/line)

D: 0=strap D out (required for block mode/page)
1=strap D in (required for block mode/line)

x: reserved

If this word is negative when OPENTERM is called, the automatic setting of the BLOCK MODE key and BLOCK MODE/PAGE straps is bypassed (the physical switch settings are used, no programmatic settings are done). If you want to use the programmatic features, you should set this word to zero or a positive value before OPENTERM is called.

The only portion of this word that is controllable by the user program is the sign bit (P, above). All the other bits are set by OPENTERM after it is called. READTERM and CLOSETERM make use of these values. OPENTERM sets the devicefile TERMTYPE (terminal type) to 10, unless it is already set to 10 or it is a multipoint terminal (terminal type 14) or Katakana terminal (terminal type 12). Upon normal termination of CLOSETERM, the ECHO, MPE SETMSG, and log-on MPE TERMTYPE values are restored if the terminal devicefile is $STDIN/$STDLIST.

TERMINAL ALLOCATION INFORMATION (WORD 4)

bits:

0 1 2 3 4	5 6 7 8 9 10 11 12 13 14 15
TYPE	SPEED

TYPE is the MPE TERMTYPE for a terminal allocated by OPENTERM. SPEED is the input/output speed (in characters per second) for a terminal allocated by OPENTERM. If TYPE or SPEED is zero when OPENTERM is called, then the default values (TYPE = 10 and SPEED = 240) are used. TYPE and SPEED are applicable only when the terminal devicefile is something other than $STDIN/ $STDLIST and :DATA has not been used. In other words, they are used only when you have specified a :FILE command using the DEV= parameter to define the terminal.

DATA OVERRUN LOGGING (WORD 5)

bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved								OVERRUN COUNT							

OVERRUN COUNT is the number of data overruns that were encountered in the previous call to READ-TERM.

READ TIMEOUT LOGGING (WORD 6)

bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved								TIMEOUT COUNT							

TIMEOUT COUNT is the number of read timeouts that were encountered in the previous call to READTERM.

OTHER DATA ERROR LOGGING (WORD 7)

bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved								DATA ERROR COUNT							

DATA ERROR COUNT is the number of recoverable errors other than data overruns and read timeouts that were encountered in the previous call to READTERM.

MAXIMUM NUMBER OF RETRIES (WORD 8)

value = 0	:	use default value (4 retries)
value > 0	:	use this value as maximum
value < 0	:	do not perform any retries

Up to the given number of data overruns, read timeouts, and other errors may occur with automatic recovery. After the last retry, the appropriate MPE file error number is returned in the STATUS word in COMMAREA. If the retry recovery attempt(s) were successful, the value returned in the STATUS word is 0 if no other errors were detected, and the number of retries are reported in COMMAREA(4), COMMAREA(5), and COMMAREA (6) (zero-origin).

SUPPRESS MESSAGES IN OPENTERM AND CLOSETERM, AND ENABLE AUTOREAD FEATURE IN READTERM (WORD 9)

bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved													R		M

M: 0=display mode messages.
1=suppress mode messages.

R: 00=suppress AUTOREAD special feature.
01=enable AUTOREAD special feature.
(This feature is used internally by DEL.)

If this value is odd (M=1) when CLOSETERM is executed, the message "REMEMBER TO UNLATCH THE BLOCK MODE KEY." is not given by CLOSETERM. If this value is even (M=0) when CLOSETERM is executed and BLOCK MODE is not cleared programmatically, the message is displayed.

If this value is odd (M=1) when OPENTERM is executed, the mode set message is not given by OPENTERM. If this value is even (M=0) when OPENTERM is executed, the message is displayed. The mode set message will be either "BLOCK MODE/PAGE IS SET." or "BLOCK MODE/LINE IS SET.".

If R is binary 01 when READTERM is executed, the AUTOREAD feature is used by READTERM; otherwise, the normal data entry mode is used. The AUTOREAD feature causes READTERM to send an "ESC d" to the terminal instead of waiting for the ENTER key to be pressed. This feature allows for performance measurements to be taken.

To invoke the default conditions, set this word to zero. The upper bits of this word are reserved for future use and should always be set to zero. Examples: 0=> M=0, R=0 . 1 => M=1, R=0. 2 => M=0, R=01. 3 => M=1, R=01.

ENVIRONMENT INFORMATION (WORD 10)

bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved													CPU		

CPU: 0=MPE-C, 1=non-MPE-C

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. The information in this table is placed in the program buffer specified by the editdef parameter when calling NEXTEDIT, ALPHAEDIT, or EDITFIELD.

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*=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 + number of fields ~~(1)~~
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

TERMINAL TYPE CODES

APPENDIX C

The termtype parameter of the :HELLO command is used by MPE to determine device-dependent characteristics such as delay factors for carriage returns. The only terminal types allowed when running FORMAINT or programs calling DEL procedures are 10, 12, and 14. If you enter any other terminal type, OPENTERM changes the termtype to 10*. Here are the definitions of these terminal types:

10 HP 2640A/B, HP 2641A, HP 2644A, HP 2645A, or HP 2648A (when used predominantly in character mode.) (10-240 cps).

12 HP 2645K (Katakana/Roman) Data Terminal.

14 Multi-point terminal.

*If you operating with MPE-C, you should specify terminal type 10. It is not programmatically set on these systems.

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

	CONTROL CHARACTERS		DISPLAYABLE CHARACTERS						ESCAPE SEQUENCES					
7	0	0	0	0	1	1	1	1	0	0	1	1	1	1
BIT 6	0	0	1	1	0	0	1	1	1	1	0	0	1	1
4321 5	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0000	NUL	D_L DLE	SP	0	@	P	`	p	SP SPACE	0 PRINT	@	P DELETE CHAR	`	p f_1
0001	S_H SOH	D_1 DC1	!	1	A	Q	a	q	!	1 SET TAB	A ↑	Q INSERT CHAR ON	a CURSOR SENSE	q f_2
0010	S_X STX	D_2 DC2	"	2	B	R	b	r	"	2 CLEAR TAB	B ↓	R INSERT CHAR OFF	b KEYBOARD ENABLE	r f_3
0011	E_X ETX	D_3 DC3	#	3	C	S	c	s	#	3	C →	S ROLL UP	c KEYBOARD DISABLE	s f_4
0100	E_T EOT	D_4 DC4	$	4	D	T	d	t	$	4	D ←	T ROLL DOWN	d ENTER	t f_5
0101	E_Q ENQ	N_K NAK	%	5	E	U	e	u	%	5	E RESET TERMINAL	U NEXT PAGE	e	u f_6
0110	A_K ACK	S_Y SYN	&	6	F	V	f	v	& PARA-METER SEQUENCE	6	F	V PREV PAGE	f	v f_7
0111	△ BEL	E_B ETB	'	7	G	W	g	w	'	7	G CURSOR RETURN	W FORMAT MODE ON	g	w f_8
1000	B_S BS	C_N CAN	(	8	H	X	h	x	(	8	H ↖	X FORMAT MODE OFF	h	x
1001	H_T HT	E_M EM	)	9	I	Y	i	y	) DEFINE SET	9	I HORI-ZONTAL TAB	Y DISPLAY FUNCTIONS ON	i	y
1010	L_F LF	S_B SUB	*	:	J	Z	j	z	*	:	J CLEAR DSPLY	Z DISPLAY FUNCTIONS OFF	j	z TEST
1011	V_T VT	E_C ESC	+	;	K	[	k	{	+	;	K ERASE TO END OF LINE	[START UNPROTECT FIELD	k	{
1100	F_F FF	F_S FS	,	<	L	\	l	|	,	<	L INSERT LINE	\	l MEMORY LOCK ON	|
1101	C_R CR	G_S GS	-	=	M	]	m	}	-	=	M DELETE LINE	] END UNPROTECT FIELD	m MEMORY LOCK OFF	}
1110	S_O SO	R_S RS	.	>	N	^	n	~	.	>	N	^ STATUS	n	~
1111	S_I SI	U_S US	/	?	O	_	o	DEL	/	?	O	_	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.

DATA ENTRY LIBRARY FORM LAYOUT SHEET

HEWLETT hp PACKARD

Page ______ of ______

FORM FILE NAME

FORM NAME

CHAIN TO FORM NAME

	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	79
0																																																																															
1																																																																															
2																																																																															
3																																																																															
4																																																																															
5																																																																															
6																																																																															
7																																																																															
8																																																																															
9																																																																															
10																																																																															
11																																																																															
12																																																																															
13																																																																															
14																																																																															
15																																																																															
16																																																																															
17																																																																															
18																																																																															
19																																																																															
20																																																																															
21																																																																															
22																																																																															
23																																																																															

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

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
								Low Value	High Value		

APPENDIX G

FCHECK ERROR CODES

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 *tcount* parameter value exceeded the *recsize* parameter, but the *multirecord access aoption* 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".
	-4	Buffer length is less than 64 bytes long.
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 2640/41/44/45/48, or terminal strapping.
	>0	Error code from "FCHECK".
	-1002	Terminal is incorrectly strapped. If connected to HP 3000 with MPE-C operating system, terminal must not be strapped for BLOCK MODE/PAGE and Terminal Mode Information in communications area must be set to negative value before OPENTERM is called. H strap may not be removed. If D strap is removed, it is recommended that G strap also be removed.
	-2006	Programmatic MPE command :SETMSG has failed.
WRITETERM	0	Operation successful.
	>0	Error code from "FCHECK".
READTERM	0	Operation successful.
	>0	Error code from "FCHECK".
	-1 to -8	-1 through -8 indicate corresponding function keys f1 through f8 have been pressed.
	-1001	Length of buffer for READTERM data is insufficient for BLOCK MODE/PAGE read. Minimum length is number of characters plus number of fields plus one. Increase buffer size or set Terminal Mode Information in communications area to negative value if BLOCK MODE/LINE desired.

Table H-1. DEL Status Word Settings (continued)

PROCEDURE	VALUE RETURNED TO STATUS WORD	MEANING
	-1003	Status request sent to 264x terminal before READTERM was called. Do not send ESC^ (terminal status request) before calling READTERM. Use TERMSTATUS to obtain terminal status information.
	-1004 X	On MPE-C operating system, requested read length exceeds 218 characters.
	-2001 X	During a BLOCK MODE/PAGE read in READTERM, a DC2 character was expected but some other character was received (264x problem).
	-2002 X	After BLOCK MODE/PAGE read in READTERM, a BLOCK TERMINATION character was expected but some other character was received (264x problem).
	-2003	After BLOCK MODE/PAGE read in READTERM, the number of FIELD SEPARATION characters received was not number expected (264x problem).
	-2004	After BLOCK MODE/PAGE read in READTERM, the number of characters received was not number expected (264x problem).
	-2005 X	During STATUS read in TERMSTATUS, a DC2 character was expected but some other character was received (264x problem).
	-2007	Invalid escape sequence received in READTERM (264x problem).
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".

INDEX