Contents

Figures

Tables

Preface

This manual describes the FMS-11 Software System for use with the RSX-11M and RSX-11M-PLUS operating systems. Overview sections tell how FMS-11 components work together. Detail sections introduce the features of each component.

This manual and FMS-11 software are primarily for RSX-11M/M-PLUS system programmers who have experience with BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV, FORTRAN-77, and MACRO-11 programs.

The Chapter Summary section briefly describes the chapters in this manual. The Symbols and Conventions section describes the documentation conventions.

Chapter Summary

Chapter 1 introduces form processing and presents general information about the FMS-11 software components.

Chapter 2 describes the FMS-11 Form Editor in detail. The Form Editor is the FMS-11 component that creates and modifies computerized form descriptions for later use in form applications.

Chapter 3 describes the FMS-11 Form Utility in detail. The Form Utility is a system utility for manipulating form descriptions by creating form library files (printable files that show how forms have been designed), object modules for forms, and directories of library files.

Chapters 4, 5, 6, and 7 introduce and describe the FMS-11 Form Driver. The Form Driver is the FMS-11 component that displays forms. It also accepts data that operators type in response to the forms.

Chapter 4 introduces Form Driver concepts in two major sections:

1. How the Form Driver interacts with the form descriptions that you have created with the Form Editor, including detailed descriptions of how the Form Driver treats field attributes and form attributes.
2. How the Form Driver interacts with the terminal operator when an FMS-11 application is running, including detailed descriptions of error handling, field editing functions, and input termination in fields and forms.

Chapter 5 details programming requirements and concepts. The first major section presents requirements applicable to all programming languages. Separate sections for BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV and FORTRAN-77, and MACRO-11 present the special details that apply only to each of those languages.

Chapter 6 is the main reference chapter for the Form Driver calls. Detailed descriptions of the calls are arranged in alphabetical order by the high-level language call name. Each description is organized as follows:

1. The purposes and effects of the call - This information applies to all programming languages.
2. The forms of the call - For each high-level language, the general forms of the CALL statement are given in full, each with an argument abbreviation that represents the purpose of the argument. For MACRO-11, the general form of the macro call to the Form Driver ($FDV) is given.
3. The input and output arguments for the call - The required and optional inputs and outputs are listed in tables that show the argument abbreviations and explain the requirement on input or the value on output. This information applies to all programming languages.
4. The codes and values for the status of the call - The high-level language status values and the MACRO-11 status codes are listed and explained in separate tables.

Chapter 7 presents programming techniques that illustrate special Form Driver capabilities and show useful combinations of Form Driver calls. Examples are included for MACRO-11 and the high-level languages.

Chapter 8 describes how to prepare your system for FMS applications. Separate sections cover the relevant RSX-11 system generation options, FMS-11 installation procedures, and the FMS-11 configuration procedure for the Form Driver, including a full listing of the interactive configuration dialog for running FMS-11 applications.

Five appendixes supplement the manual. Appendix A contains a listing of the FMS-11 system macro library (FMSMAC.MAC). Appendix B contains FMS sample forms. Appendix C contains source listings of executable examples of FMS-11 that DIGITAL has distributed as part of your kit. Appendix D contains source listings of task building FMS sample applications. Appendix E lists and explains all of the FMS-11 error messages.

Symbols and Conventions

This manual and the *FMS-11/RSX Mini-Reference* use the following symbols and conventions. Although most of them are the same as the symbols and conventions used in other documents for PDP-11 software, the VT200 terminal and the video orientation of the keypad editor require a few changes.

System prompts	Indicate the system is ready for your command. System prompts are: For RSX-11M systems: the characters `MCR>` or a right-angle bracket (`>`) by itself. For RSX-11M-PLUS systems: the dollar sign (`$`) or the RSX-11M prompts (when the MCR is running).
Red print	Indicates characters typed on the keyboard in examples.
Black print	Indicates the characters displayed by the system or by the FMS-11 software in examples.
Uppercase letters (In commands and calls)	Indicate the characters you must type on the keyboard (see also "Lowercase letters").
Lowercase letters (In commands and calls)	Indicate the parts of commands or command strings you must supply (see also "Uppercase letters").
CTRL/	Indicates a combination of the control key and another keyboard key. For example, for CTRL/U hold down the CTRL key and press the U key.
Square brackets ([ ]) (In general forms or commands)	Enclose an optional term or optional characters (do not type square brackets as part of a command unless the instructions explicitly require them).
Braces ({ })	Enclose a list of two or more terms from which you must choose and type one (do not type braces as part of a command).
Dot matrix letters	Indicate prompts, short status messages, and examples. .end;

Chapter 1
Introduction to FMS-11

1.1 Overview

FMS-11 is DIGITAL's Form Management System. FMS-11 software contains the tools for developing form applications and running them on VT200, VT100, and VT52 terminals. In the past, printed forms have been the most common tool for collecting and transmitting data in an orderly manner. FMS-11 software now brings the speed, convenience, accuracy and low cost of computerized processing to users who have been using printed forms.

The FMS-11 software described in this manual is designed to run on RSX-11M and RSX-11M-PLUS systems for user application development and execution. In addition, many FMS-11 application programs developed for RSX-11M or RSX-11M-PLUS systems can be executed on RSX-11S Version 4.2.

Forms are designed by typing them directly onto the terminal screen. Neither layout charts nor a special forms design language is required. FMS-11 associates constant data with the form, not with the application program. This makes for simplified application program maintenance and increased application program flexibility. You can modify forms later without recompiling your application program.

Form application programs can be written in one of several programming languages. FMS-11 provides language support for BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV, FORTRAN-77, and MACRO-11.

FMS-11 software contains three main components for developing and executing form application programs:

- The Form Editor (FED)
- The Form Utility (FUT)
- The Form Driver (FDV)

1.1.1 The Form Editor

The Form Editor (FED) simplifies designing, modifying, and storing form descriptions for video display. Your screen always displays your form in its current state. Keypad and keyboard functions allow you to specify video display characteristics for either constant text or fields that contain picture characters. In the form descriptions, you can include short, helpful explanations about individual fields or about each form as a whole. This will help future operators understand the purpose of each form.

When you design forms, you assign form names and field names and you refer to named data that will be used (but not displayed) by the Form Driver when the form is used by an application task. The actual design of the form and the specific application task requirement control the desired operator response to information displayed or data to be entered on forms.

1.1.2 The Form Utility

The Form Utility (FUT) allows you to create versions of form descriptions that are suitable for hardcopy listings, to create and modify form libraries, to list the names of forms contained in a form library, and to produce object modules of form descriptions. You can task build the object modules with form application code to produce form applications that are entirely memory-resident. The Form Utility also generates COBOL data division code suitable for copying into a COBOL program to correspond to a form definition.

1.1.3 The Form Driver

The Form Driver (FDV) is a set of subroutines that permit your application program to access forms you created by using the Form Editor. Application programs access forms by issuing Form Driver calls that are embedded in the program and are written in the source language of the program. All Form Driver calls refer to specific forms, or fields within forms, by using names that you assign during the form editing process. The Form Driver performs field and character validation for operator input based on the form description (validation is based on picture-validation characters and field attributes). The Form Driver also responds to operator HELP requests by displaying appropriate help text associated with the form and field being processed.

1.2 Developing Form Applications

Seven stages comprise the typical development cycle for form application programs:

- PLAN

  Study the existing process that the FMS-11 application will improve; list the data that operators can provide; list the hardware resources that operator sites will have; describe the current skills and the additional skills they will need; specify the features that FMS-11 forms for the application are to have and the processes that the form application programs are to perform.

- DESIGN FORMS

  Use the Form Editor to lay out and modify the forms that the form application programs will use. Use the Form Utility to print form descriptions for reference, to create object modules for form descriptions that are to be memory resident, to store forms in a form library file, and to list the names of forms in a form library file.

- WRITE TASKS

  Use the Form Driver calls in the form application program to process form descriptions, to handle form-related terminal I/O, and, to a limited extent, to check the validity of operator responses.

- DEBUG TASKS WITH FORMS

  Confirm that all processes using the application's forms work correctly.

- VALIDATE ON OPERATOR SITE SYSTEMS

  Confirm that the forms and application software work correctly on each type of target system on which they will be used.

- PREPARE APPLICATION SYSTEM DOCUMENTATION

  Provide complete documentation for operators who will use the FMS-11 forms and application program software.

- DISTRIBUTE

  Package and distribute the FMS-11 forms, application program software, and user documentation as a complete application system package.

The two major steps required when developing form applications are designing forms and writing application programs. Chapter 2 contains the information that you will need to design and modify forms. After forms have been designed, the Form Utility allows you to create and maintain form library files. The Form Utility is described in detail in Chapter 3.

The application program writing stage deals with the use of the Form Driver. Details for writing Form Driver application programs are contained in Chapters 4 through 7. These chapters include information on Form Driver interaction between forms and the operator as controlled by Form Driver calls issued by the application program, application programming requirements and concepts, Form Driver calls, programming techniques and examples, and building and running form application programs.

Chapter 2
The FMS-11 Form Editor (FED)

2.1 Overview

The FMS-11 Form Editor (FED) allows you to create, modify, and store customized forms on a VT200 or VT100 terminal. Your application programs use these forms to collect data entered by an operator at a video terminal.

Creating or editing a form with the Form Editor is an interactive and iterative process. You need not know in advance all the details or all the modifications you intend to specify for a form. The Form Editor allows you to test various possibilities, observe their appearance on the screen, and choose the design you consider most successful.

The product of your work with the Form Editor is a form description that can be saved in a file or form library and be retrieved from the file or form library for additions or changes. You can change individual fields or text portions of the form without affecting other fields or text. For example, you might want to reposition items on the screen to make the form more attractive to the eye or easier for an operator to handle, add or remove fields, or supply additional HELP text. You can make these and other changes by starting the Form Editor, editing the screen image of the form to make the desired changes in the form description, and saving the modified form description in the file or form library.

The purpose of the form description is to provide information to another software component called the Form Driver. The Form Driver handles the interaction of the terminal operator with the form displayed on the screen and with the application program. The Form Driver is described in Chapter 4.

In summary, the Form Editor allows you to perform these operations:

1. Creation and modification of a form's screen image by means of the terminal's main keyboard and the text editor keypad.
2. Storage and retrieval of form descriptions from form files or libraries.

2.2 Form Editor Terminology

2.2.1 Screen Form

The screen form is a video display that resembles a paper form. The computer creates the display from a form description that specifies to the computer the proper characters to display on the screen.

2.2.2 Form Description

The form description is the computer's specification of a screen form. It indicates which characters to display on the screen as well as the location, size, and other characteristics of each field. The specification also includes the name of the form and how the form and its fields are processed.

2.2.3 Field

A field is a set of contiguous characters (either picture-validation or field-marker) terminated by a blank, a non-field character, an end-of-line delimiter, or a change in video attributes. A field can be left blank for some of the information a form is designed to use.

2.2.4 Form Description File

The form description file is a computer file containing only one form description that might or might not be complete or accurate. It is a binary file that has been arranged so FMS can use it to display screen forms.

2.2.5 Form Library File

The form library file is a computer file containing at least one form description and a directory of the names for each form description. It is a binary file but is arranged so that individual form descriptions can be taken out by name for use by FMS.

2.3 Starting the Form Editor

The Form Editor (FED) requires a VT200 or VT100 terminal. The terminal must be made known to the system as a VT200 or VT100.

RSX-11M/M-PLUS system users can use the MCR SET command for this:

```
SET /VT200=ti:
```

The *RSX MCR Operations Manual* describes the SET command.

If you are using your VT200 in 8-bit mode, you must make sure you have set the terminal characteristic to 8-bit with the following command:

```
SET TERM /EIGHT_BIT
```

If you do not set this terminal characteristic, you might receive erroneous results.

The Form Editor requires the full-duplex terminal driver to run.

The main keyboard performs normally when you are using the Form Editor, allowing you to insert characters, delete them, etc. The keypad to the right of the keyboard provides operations specifically related to the Form Editor.

The following operations are used to design a form:

- **Start the Form Editor**

  By using the standard commands to load the Form Editor into memory, you begin program execution.

  When the Form Editor prompt (`FED>`) is displayed on the screen, you can type in a response. The response describes the form file you want to create or edit, or the library that contains the desired form. The response requires a prescribed syntax, which will be described in the "File Specification" section.

  To start the Form Editor if it is in the system account, type:

  ```
  RUN $FED RET
  ```

  The Form Editor clears the screen, displays the prompt `FED>` at the bottom of the screen, and accepts a command line.

  If the Form Editor is installed on your system, you can start it by typing:

  ```
  FED RET
  ```

  The Form Editor is built with buffer space that should be sufficient to edit almost any form. It will allow you to create or edit a form description of up to 2048 words.

  However, if you are editing forms larger than 2048 words or are editing forms smaller than that, you can change the size of the FED task. FED can be installed or run with a different task increment for RSX-11M and 11M-PLUS systems:

  ```
  INS $FED/INC=words
  ```

  or

  ```
  UN $FED/INC=words
  ```

  The default task extension FED is built with 4096 words. The minimum extension required for FED to run is 1024 words.

Figure 2-1 shows examples of commands that allow you to get into the Form Editor, create a new form, find the version number of the Form Editor, load an old form, and exit the FED utility.

```
>RUN $FED        (get into Form Editor)

FED>  /CR        (create a new form)

FED>  /ID        (find version number of FED)

FED>  formname   (load an old form)

FED>  <ctrl>-Z   (exit Form Editor utility)
```

Figure 2-1. Form Editor Commands

- Make File Specifications

  The output of the Form Editor always goes to a form file. Each form file contains only one form description. (To create or update form libraries, use the Form Utility (FUT), described in Chapter 3.)

  To create a new form description, use the /CR option:

  ```
  FED>/CR  RET
  ```

  To edit a form description contained in a form file, type the name of the form file, for example, "VENDOR".

  ```
  FED>VENDOR.FRM  RET
  ```

  To extract a form description from a form library for editing, type the name of the library file and respond to the FED prompt `Form name?` with the name of the form. The default file type is .FRM, indicating a form file. If the file type is .FLB, you must type .FLB explicitly:

  ```
  FED>DEMLIB.FLB  RET

  Form name? FIRST  RET
  ```

  If the specified form is not found, FED repeats the `Form name?` prompt. If you press the RETURN (RET) key in response to the `Form name?` prompt, FED displays the `FED>` prompt again and waits for a new command line.

  It isn't necessary to distinguish between a form file and a library file on the command line. FED determines whether the input file is a form file or a library file, and proceeds accordingly.

  If you want to know the version number of the Form Editor, display its identification message by typing:

  ```
  FED>/ID  RET
  ```

File specification formats may differ according to the operating system you use.

The syntax for the RSX-11M/M-PLUS system form file and library file specifications is:

```
dev:[UIC]filename.type;version
```

If the device is not specified for the input file, FED assumes the system device. For RSX systems, if the User Identification Code (UIC) is omitted, FED defaults to the currently assigned UIC.

The default file type for the input file is .FRM (a form file). If an explicit version number is not specified for an input file, FED uses the latest version of the file.

The output file that FED creates during an editing session is always a form file with the file specification "form.FRM". "Form" is the name of the form when the session ends. On RSX-11M and RSX-11M-PLUS systems, FED creates the output file on the system device in the RSX account under which the Form Editor is running.

- **Issue Form Editor Commands**

  You can use any of the several commands summarized in Table 2-1 to enter a particular phase of the Form Editor. You type the commands in response to the `COMMAND:` prompt. If you type HELP in response to the `COMMAND:` prompt, the Form Editor displays the valid responses to the prompt.

Table 2-1. FED Command Summary

Command	Abbreviated Command	Function
HELP	H	List the commands available in the Form Editor.
EDIT	ED	Create or edit the form's screen image.
ASSIGN *[option]*	A *[option]*	Assign field attributes. ("Attributes" are characteristics of fields that you assign with FED for use by the Form Driver.)
where *option* can be one of the following:		
ALL	A	Assign attributes for old and new fields.
NEW	N	Assign attributes for changed or newly created fields.
FIELD *nam*	F nam	Assign attributes for the field called nam.
FORM	F	Assign form-wide attributes. (Form-wide attributes apply to an entire form rather than to a particular field.)
NAME	N	Enter and edit named data. (Named data is information that is to be associated with a form, but not displayed with it.)
SAVE	None	Store the form, and return to the `FED>` prompt. Both input and output files are preserved.
QUIT	None	Cancel a session without saving output files, and return to the `FED>` prompt. The input file is preserved.

- **Use Keypad Operations**

  The keypad layout for the Form Editor in Figure 2-2 shows the operations that are associated with certain keys or key combinations.

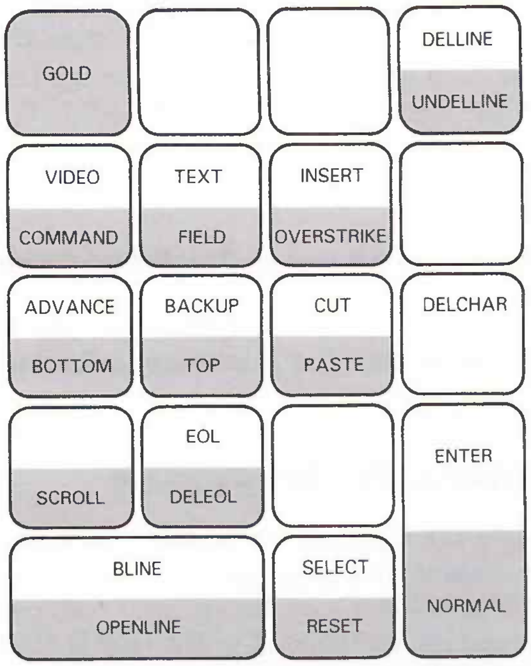


Figure 2-2. Form Editor Keypad Layout

2.4 Form Editor Commands

You can type any one of the commands shown in Table 2-1, FED Command Summary, in response to the `COMMAND:` prompt. The Form Editor enters the specified command after you press the ENTER key on the keypad. If you want to cancel your last command, type the CTRL/U combination before you press the ENTER key. If you want to change a command, use the DEL key to delete the characters that make up the command. You can use CTRL/U to delete the entire command line and then type in a new command.

You can type the HELP command to display a list of the Form Editor commands and their functions.

Begin an editing session with a rough pencil sketch of the form you want to create. You can perfect the details of the form interactively with the Form Editor by looping back through the command functions (using the GOLD/COMMAND key sequence) and adding or deleting features gradually during the development of the form design.

During a form editing session, the various command operations let you move in an orderly manner from one phase of your work to another, allowing you to control the phases of your work. You can enter any phase at any time. The FORM, ASSIGN, and NAME phases use the Form Driver to display and collect responses with questionnaire forms.

Form Driver key operations are active while you are completing any of the questionnaires. For example, when you are assigning form-wide, field, and named data attributes, the TAB key has the effect of moving the cursor to the first character position of the next field, and the BACKSPACE key moves the cursor to the previous field. Chapter 4, Introduction to the FMS-11 Form Driver (FDU), describes Form Driver key operations in detail.

2.4.1 Assigning the Form-Wide Attributes: The FORM Command

The FORM command displays the Form-Wide Attribute Questionnaire (Figure 2-3). The operator enters the necessary information into the questionnaire to create a form file.

2.4.2 Editing the Form Display: The EDIT Command

The EDIT command causes the Form Editor to enter the edit phase. During this phase, the operator creates and modifies a screen image of the form. During the edit phase, it is possible to type background text, create fields and scrolled areas, and assign certain attributes. Use the GOLD/COMMAND key sequence to return to the `COMMAND:` prompt.

2.4.3 Assigning Field Attributes: The ASSIGN Commands

The ASSIGN command with any of its options tells the Form Editor to enter the field attribute assignment phase. For a new form, the ASSIGN command is often used after the completion of the edit phase. The Form Editor displays the Field Attributes Questionnaire (Figure 2-4) that requests field attributes for each field in the form. For an existing form, the operator need only enter field attributes not assigned earlier.

It is possible, during the ASSIGN phase, to exit before completion of all field attribute assignments. To do this, press the period (.) key on the keypad. This will cause a return to the `COMMAND:` prompt and assign default attributes to all remaining fields.

2.4.3.1 For All Fields: The ASSIGN ALL Command — Causes the Form Editor to request attributes for all fields. To display the questionnaire for the next field, press the ENTER key.

2.4.3.2 For New and Changed Fields Only: The ASSIGN NEW Command — Causes the Form Editor to request attributes for new fields only. To display the questionnaire for the next field, press the ENTER key.

2.4.3.3 For a Specified Field Only: The ASSIGN FIELD Command — This must be followed by a field name. It allows the operator to assign attributes to a particular field.

2.4.4 Specifying the Named Data: The NAME Command

The NAME command causes the Form Editor to enter the named data assignment phase. The Form Editor displays a questionnaire that collects names and data to be associated with those names. Named data is used typically to hold information about a form in the form description, but outside the form itself. Named data is not displayed with a form.

2.4.5 Storing the Form Description: The SAVE Command

SAVE causes the Form Editor to place the present form description in the output file and return to either the `FED>` or system prompt, depending on how the program began.

If field attributes have not been assigned to all fields when the SAVE operation is performed, the Form Editor supplies default values for any fields whose attributes have been left unspecified; a default name of all blanks is supplied as the field name.

2.4.6 Canceling the Session Without Saving the Form: The QUIT Command

The QUIT command causes the Form Editor to return to the `FED>` or system prompt without saving the current form in an output file; this form is deleted.

2.5 Edit Status Display

During the EDIT phase, the bottom line (Line 24) of the screen displays information about the current status of the Form Editor. The format for the line is:

```
CURSOR: TXT NOR  LIN 1  COL 1   MODES: TXT ADV INS  SELECT: LIN 1 COL 1
        FLD SCR      23    132         FLD BCK OVS              23   132
```

The second line (above) indicates the alternative choice or the limitations of the items in the display.

The fields on Line 24 are displayed in reverse video.

The following list includes each status and its description.

CURSOR	This section indicates the cursor line and character locations.
`TXT/FLD`	The cursor character is either a text (`TXT`) character or a field (`FLD`) character.
`NOR/SCR`	The cursor line is either a normal screen line (`NOR`) or a part of a scrolled region (`SCR`).
`LIN 1-23`	The line number at which the cursor is located.
`COL 1-132`	The column number at which the cursor is located.

MODES	This section indicates the status of the internal mode indicators of the editor.
`TXT/FLD`	The current input mode is either text (`TXT`) or field (`FLD`).
`OVS/INS`	The current input mode is either overstrike (`OVS`) or insert (`INS`).
`ADV/BCK`	The current direction for move operations is either advance (`ADV`) or backup (`BCK`).
SELECT	This section is present only if a select range is active. Otherwise, this portion of the line is blank.
`LIN 1-23`	The line number at which the select point is located.
`COL 1-132`	The column number at which the select point is located.

2.6 Form Editor Operations Reference

This section describes the creation of the form's screen image during the EDIT phase and the assignment of all attributes during the FORM, EDIT, ASSIGN, and NAME phases.

2.6.1 Creating the Form's Screen Image

The Form Editor includes a text editor for creating and modifying screen images. The text editor lets you use standard operations for mode-changing, cursor control, and text modification.

The Form Editor lets you define fields and background text in the form for data input/output between your application and the terminal operator. It also enables you to assign video attributes (such as bold, blink, and underline) to any character or set of characters on the terminal screen, and to define a block of lines as a scrolled area.

NOTE

> To run the Form Editor, you need a VT100 or VT200 terminal. Neither FED nor FDV supports the double-high or double-wide video attributes.

2.6.2 The Text Editor

The keyboard performs like a typewriter when you use the Form Editor; it lets you input and delete characters. The keypad to the right of the keyboard provides operations specifically related to the Form Editor. It is recommended that you make a copy of the keypad layout and keep it at the terminal.

This keypad provides four kinds of operations:

- Mode-Changing Operations

  You change modes by pressing the appropriate key or key combination on the keypad. Modes determine placement of characters, movement forward or backward through the form, and definition of fields and background text.

- Cursor Control Operations

  These operations change the cursor position but do not affect the text. The cursor can advance only to the margin boundaries.

- Text Modification Operations

  These operations insert, delete, and modify text.

- Scroll Operation

  This operation permits the definition of a scrolled line. Together with identical lines that immediately follow it, the line becomes a scrolled area.

2.6.3 Mode-Changing Operations

The Form Editor works in several modes. The mode choices are TEXT/FIELD, OVERSTRIKE/INSERT, and ADVANCE/BACKUP. Only one of each pair can be active at one time.

The TEXT/FIELD modes tell the Form Editor whether the characters you enter are background text characters for the form (TEXT mode) or the special set of field characters that define the picture format of a field (FIELD mode). The special set of field characters includes field-marker (such as slashes and hyphens that delimit fields) and picture-validation characters.

The OVERSTRIKE/INSERT modes determine how the Form Editor places characters in the form with respect to characters already there.

The ADVANCE/BACKUP modes determine whether the Form Editor executes an operation in a forward (right and down) or backward (left and up) direction.

2.6.3.1 TEXT/FIELD — TEXT mode is activated by pressing the TEXT key on the keypad. FIELD mode is activated by pressing the GOLD/FIELD key sequence. In TEXT mode, the Form Editor accepts any character as input. It enters any printable character or space in the background text of the form. The Form Driver does not see these characters as data. Rather, it treats the characters as constant text that is always displayed on the form. TEXT mode is deactivated by pressing the GOLD/FIELD key sequence, which places you in FIELD mode.

In FIELD mode, the Form Editor accepts as input only the picture-validation characters A, C, N, X, and the digit 9, as well as a set of ASCII field-marker characters. Picture-validation characters tell the Form Editor whether to accept alphabetic (A), alphanumeric (C), numeric (9), signed numeric (N), or any characters (X) as input for each character position in a field. Field-marker characters are text characters that you can define as part of a field. Valid field-marker characters include the hyphen (-), slash (/), asterisk (*), dollar sign ($), pound sign (#), and comma (,).

If, while in FIELD mode, you enter a character that is neither a field-marker nor a picture-validation character, the Form Editor sounds the terminal bell and rejects the input. The Form Editor accepts a blank as input in FIELD mode, but does not make it part of the field. Field-marker and picture-validation characters are treated as such only when the Form Editor is explicitly in FIELD mode. For example, the digit 9 is associated with a field as a picture-validation character if it is typed in FIELD mode; otherwise, it is treated as a text character.

You can deactivate FIELD mode and return to TEXT mode by pressing the TEXT key.

2.6.3.2 ADVANCE/BACKUP — The ADVANCE/BACKUP modes affect the beginning of line (BLINE) and end of line (EOL) operations. They do not affect character insertion or deletion.

ADVANCE mode causes the Form Editor to implement operations in the direction moving from the current cursor position toward the end of the line or form. You can deactivate ADVANCE mode by pressing the BACKUP key.

BACKUP mode causes the Form Editor to implement operations in the direction toward the beginning of the line or form. You can deactivate BACKUP mode by pressing the ADVANCE key.

2.6.3.3 INSERT/OVERSTRIKE — The INSERT/OVERSTRIKE modes affect the way characters are placed or moved when you type or make deletions.

INSERT mode places typed characters at the current cursor location and moves the cursor to the right. Any other characters on the line are moved over to make room for the inserted character. If characters would be lost by being pushed beyond the margin, the Form Editor sounds the terminal bell and rejects the insertion.

If you delete a character in INSERT mode, the Form Editor removes the character to the left of the cursor and characters to the right slide over to close the space.

Deactivate INSERT mode by pressing the GOLD/OVERSTRIKE key sequence.

OVERSTRIKE mode causes the Form Editor to replace the character at the current cursor position with the new character typed at the terminal. When a character is deleted, adjacent characters do not close up the line. The character is erased. The deleted character is replaced by a blank, and the cursor is positioned on that character's space. You can enter OVERSTRIKE mode by pressing the GOLD/OVERSTRIKE key sequence.

Deactivate OVERSTRIKE mode by pressing the INSERT key.

2.6.4 Cursor Control Operations

The following operations change the cursor's position during an editing session.

The cursor symbol (either a solid rectangle or an underline) blinks on the character cursor location. A row-column counter in the lower right-hand corner of the screen displays the precise character position where the cursor symbol is blinking.

Uparrow (↑)	Press the UPARROW key once to move the cursor up one line. If you attempt to move the cursor above the top margin of the form, the Form Editor will not allow you to do so and the terminal bell will sound.
Downarrow (↓)	Press the DOWNARROW key once to move the cursor down one line. If you attempt to move the cursor below the bottom margin of the form, the Form Editor will not allow you to do so and the terminal bell will sound.
Rightarrow (→)	Press the RIGHTARROW key once to move the cursor one character position to the right. If you attempt to move the cursor beyond the right margin, the Form Editor will not allow you to do so and the terminal bell will sound.
Leftarrow (←)	Press the LEFTARROW key once to move the cursor one character position to the left. If you attempt to move the cursor beyond the left margin, the Form Editor will not allow you to do so and the terminal bell will sound.
BLINE	Press the BLINE key to move the cursor to the beginning of a line. The cursor will move either forward or backward, depending on whether the Form Editor is in ADVANCE or BACKUP mode when you press the BLINE key. If the Form Editor is in ADVANCE mode, BLINE moves the cursor to the beginning of the next line. Pressing BLINE again moves the cursor to the beginning of the subsequent line.

	If the Form Editor is in BACKUP mode, BLINE moves the cursor to the beginning of the current line. Pressing BLINE again moves the cursor to the beginning of the previous line.
	If you attempt to move the cursor to a line beyond the top or bottom screen boundary, the Form Editor will not allow you to do so and the terminal bell will sound.
RETURN	The RETURN key on the keyboard is equivalent to BLINE when the Form Editor is in ADVANCE mode. Pressing RETURN moves the cursor to the beginning of the next line.
EOL	Pressing the EOL key moves the cursor to the end of a line. The cursor will move either forward or backward, depending on whether the Form Editor is in ADVANCE or BACKUP mode when you press the EOL key.
	If the Form Editor is in ADVANCE mode, EOL moves the cursor to the end of the current line. If you press EOL again, the cursor moves to the end of the next line.
	If the Form Editor is in BACKUP mode, EOL moves the cursor to the end of the previous line.
BOTTOM	Pressing the GOLD/BOTTOM key sequence on the keypad moves the cursor to the bottom-right corner of the screen.
TOP	Pressing the GOLD/TOP key sequence on the keypad moves the cursor to the top-left corner of the screen.
REPEAT	If you press the GOLD key, a number, and an operation that you want to perform, the Form Editor repeats that operation the number of times you have specified. After you type the first digit of the number, you see the prompt `REPEAT:` and the number itself on the screen. The first command or key typed after the digits is repeated that number of times. You can edit the number using the DELETE key to increase or decrease the repetitions.

2.6.5 Text Modification Operations

Text modification operations allow you to insert, modify, and delete characters and lines in the form, as well as to assign video attributes to background text and fields.

2.6.5.1 Inserting ASCII Characters — When you type any ASCII character, the Form Editor inserts that character at the current cursor location and moves the cursor one character position to the right.

If you type a character at the end of a line, the Form Editor inserts the character in the last available position, sounds the terminal bell, and causes the cursor to "bounce back," leaving the cursor symbol at the last character position on the line.

The Form Editor handles typed characters differently depending on whether INSERT or OVERSTRIKE mode is in effect.

2.6.5.2 Inserting Characters in INSERT Mode — In INSERT mode, the Form Editor inserts the character at the current cursor position. The character previously located there moves one character position to the right. All other characters on the line to the right of the cursor move one character position to the right. If the last character on the line is not a blank, the Form Editor rejects any operation that would cause that character to be lost by pushing it off the end of the line. If any fields are moved on a line, the Form Editor automatically updates their field descriptors in the form description to reflect the change in the field's screen location.

Press the DELETE or DEL key on the keyboard to delete the character to the left of the cursor. If the cursor position is column 1 when this key is pressed, the Form Editor rejects the operation and sounds the terminal bell.

If the Form Editor is in INSERT mode, DELETE moves the cursor and the remaining characters on the line one character position to the left. A blank is inserted at the end of the line.

2.6.5.3 Inserting Characters in OVERSTRIKE Mode — In OVERSTRIKE mode, the Form Editor replaces the character at the current cursor position with the character that is typed.

Press the DELETE or DEL key on the keyboard to delete the character to the left of the cursor. If the cursor position is Column 1 when this key is pressed, the Form Editor rejects the operation and sounds the terminal bell.

If the Form Editor is in INSERT mode, DELETE moves the cursor and the remaining characters on the line one character position to the left. A blank is inserted at the end of the line.

If the Form Editor is in OVERSTRIKE mode, DELETE replaces the character to the left of the cursor with a blank and moves the cursor one character position to the left. If a field's position is changed, the corresponding descriptor is updated. However, if a field's picture is modified, it is a new field and old attributes are lost.

2.6.5.4 DELCHAR — Press the Delete Character (DELCHAR) key on the keypad to delete the character at the current cursor position. If a field's position is changed, the descriptor is updated. However, if a field's picture is changed, it is a new field and the old attributes are lost.

If the Form Editor is in INSERT mode, DELCHAR deletes the character, moves the remaining characters on the line one position to the left, and inserts a blank at the end of the line. The cursor remains in its current position.

If the Form Editor is in OVERSTRIKE mode, DELCHAR replaces the character on which the cursor is positioned with a blank and moves the cursor one position to the right. This is equivalent to typing a blank while in OVERSTRIKE mode. If the cursor is on the last character position on the line, the Form Editor deletes the character, sounds the terminal bell, and leaves the cursor in its current position. The Form Editor updates the field descriptors of fields affected by the change.

2.6.5.5 OPENLINE — Press the OPENLINE key to insert a blank line before the current line and move all remaining lines down one line. The Form Editor reassigns screen locations on the form to affected fields that already have field descriptors. If the next to last line on the screen (the last line available for your form) is not blank, the Form Editor rejects the OPENLINE operation, sounds the terminal bell, and prints an error message.

2.6.5.6 CTRL/W — Press the CTRL/W combination to redisplay the current screen and restore the keypad to application mode. This command is useful when there are power failures, static problems, or distortions.

2.6.5.7 CTRL/U — Press the CTRL/U combination to delete all characters between the current cursor position and the beginning of the line. The cursor remains at its current position.

2.6.5.8 DELEOL — Press the Delete to End of Line (DELEOL) key on the keypad to delete all characters between the cursor location and the end of the line including the cursor location, replacing them with blanks. The cursor remains at its current position.

2.6.5.9 DELLINE — Press the Delete Line (DELLINE) key to delete the current line, move all the lines below it up one line, and insert a blank line at the bottom. The Form Editor updates the field descriptors of any affected fields. The entire line is deleted regardless of the cursor position in the line.

2.6.5.10 UNDELLINE — Press the Undelete Line (UNDELLINE) key to restore the last line or line segment that you have just deleted. This operation saves you from mistaken or accidental deletions. It also provides you with an easy way to duplicate lines. For example, UNDELLINE can be used to create many identical lines in a scrolled area.

The effect of the UNDELLINE operation depends on how the original deletion was performed.

If the deletion was performed by using a DELEOL or CTRL/U, the Form Editor places the contents of the buffer containing the deleted characters at a position starting at the current cursor location. If deleted by CTRL/U, the characters are placed to the left of the cursor location; if by DELEOL, they are placed to the right of the cursor location. This restoration can be performed only if the deleted characters will be replacing blanks. Field descriptors for the original fields are restored only if the cursor remains at the location where the original deletion was made.

If the deletion was performed with DELLINE, the Form Editor performs an OPENLINE operation at the current cursor position. The deleted line is replaced on the screen in the blank line created by OPENLINE. The Form Editor updates all old field descriptors for fields affected by the OPENLINE operation when the field's position changes, but not the picture. The field descriptors for the deleted line are restored only when UNDELLINE is performed the first time and on the same line where the deletion was done.

2.6.5.11 REPEAT — If you press the GOLD key, a number, and the operation you want to perform, the Form Editor repeats that operation the number of times you have specified. After you type the first digit of the number, the prompt `REPEAT:` and the number appear on the screen. The first command or key typed after the digits is repeated that number of times. You can edit the number to increase or decrease the repetitions by using the DEL and CTRL/U operations. DEL is not a repeatable function.

2.6.5.12 SELECT — Press the SELECT key to mark the current cursor position as a reference point for video attribute assignment and CUT operations. SELECT defines the first character of the select range. The end of the select range is the final position to which the cursor has been moved. In other words, the select range is the area defined by the place where SELECT was pressed and the current cursor position. SELECT is used with the CUT, PASTE, and VIDEO operations.

2.6.5.13 CUT — Pressing the CUT key saves all the characters contained in the current select range. The characters are stored in a buffer, and blanks replace the contents of the area in the screen image. If a SELECT operation has not been performed, the Form Editor sounds the terminal bell in response to an attempted CUT.

2.6.5.14 PASTE — The PASTE operation inserts the characters saved by the previous CUT operation into the same area relative to the current location of the cursor as obtained when the original CUT operation occurred. The PASTE operation makes sure the inserted material does not cross boundary lines or any other text or fields in the form. If boundary lines are crossed, the Form Editor displays the error message "Cannot paste over margins or non-blanks or in scrolled areas" and sounds the terminal bell.

The PASTE operation is allowed only if the target paste area consists entirely of blanks. If the target paste area is not blank, the target area is painted in reverse video, and a message is displayed on line 24. When this occurs, press any key to remove the reverse video attribute, move the cursor to define a proper target area, and continue the operation.

2.6.5.15 VIDEO — Press the VIDEO key to activate video attribute assignment. The prompt `VIDEO:` appears on the terminal screen. Type any of the following responses to assign the specified attribute within the select range. Press the ENTER key after typing the response. The abbreviations are separated from the rest of the word by an asterisk (*).

Bo*ld	Displays all characters within the select range in boldface.
Bl*ink	Displays all characters within the select range in alternately increasing and decreasing screen brightness.
Re*verse	Displays all characters within the select range on a reverse screen background. If the screen is white-on-black, characters in reverse video appear in black-on-white; if the screen is black-on-white, characters in reverse video appear in white-on-black.
Un*derline	Underlines all characters within the select range.
Cl*ear	Deactivates or clears all the active video attributes in the select range.
Ed*it	This is not an attribute, but returns you to the normal screen editing mode.

You must use the SELECT operation to delimit the characters affected. The SELECT range includes both text and fields; it can cut a field in the middle, creating two separate fields if the two parts of the field receive different video attributes.

You can assign video attributes in either TEXT or FIELD mode.

Since you can use the CLEAR attribute to cancel the other video attributes, you can easily experiment with the various attributes to achieve the best effect. When you have the combination of attributes that you want to keep in your form, end the video attribute assignment session by typing EDIT or pressing the RETURN key.

Note that a character can have more than one video attribute. For example, the character can appear on the operator's screen as both bold and blinking. However, all characters in a field must have the same video attributes.

2.6.6 Scroll Operation

FMS limits the amount of information that can appear on the terminal screen at one time; the scrolling operation enables you to define sections within a form for displaying portions of large data tables. A data table is considered scrolled because you can "roll" it upward or downward to display the lines that you want the operator to see or work on. A scrolled area is a window into a form, displaying a relatively large amount of data a few lines at a time.

A scrolled area can be as small as two lines. Within one form, you can define as many separate scrolled areas as will fit within 23 lines. Each line can have as many separate fields as will fit on one screen line. Within each scrolled area, however, all lines must be identical with respect to the number, size, and attributes of fields and all other details.

Because the Form Driver can store field values only for the fields that are on the terminal screen, your program must maintain all scrolled area field values that are not displayed — all the values that are "above" and "below" each scrolled area. When your program scrolls up or down in a scrolled area, the program must collect the lines of values scrolled out of the area and display any line of values scrolled into the area.

The GOLD/SCROLL key sequence tells the Form Editor to define the current line as scrolled.

The GOLD/NORMAL key sequence removes the scrolling attribute from a line.

Once you have defined a line as scrolled, you can extend the scrolled area by using the DELLINE and UNDELLINE operations. Delete the scrolled line and then undelete (or restore) it as many times as you wish. In this way, you can be sure that the lines of the scrolled area are identical.

The GOLD/SCROLL key sequence only defines the current line as scrolled. The succeeding lines that are identical to the scrolled line are processed as part of the scrolled area. The first line that differs in any detail from the original scrolled line causes the Form Editor to terminate the scrolled area.

A scrolled area should not contain text except for field-marker characters. Once the text scrolls off the screen, it is lost.

In the field attribute assignment phase, the Form Editor asks about the fields on the first line of a scrolled area only. Fields on subsequent lines of the scrolled area are considered to have the same attributes as the fields on the first line. A form can contain more than one scrolled area.

2.6.7 Field Pictures

A field is a set of contiguous field definition characters (picture-validation or field-marker characters) terminated by a blank, a non-field character, an end-of-line delimiter, or a change in video attributes. Picture-validation attributes apply only to characters in fields. They tell the Form Driver whether the operator may input a number, a letter, etc., in response to a given field.

The Form Editor recognizes the five picture-validation characters shown in Table 2-2.

Table 2-2. FED Picture-Validation Characters

Character	Type
C	Alphanumeric
A	Alphabetic
9	Numeric
N	Signed Numeric
X	Any Character

2.6.7.1 Alphanumeric Characters (C) — The C in any character position defines what is valid input in that position. The C character is a character attribute rather than a field attribute. The C character allows the operator to input the digits 0 through 9, the letters A through Z (either in uppercase or lowercase), 8-bit characters with octal values from 300 through 375, and a space. Any other attempted input sounds the terminal bell and causes an error message.

2.6.7.2 Letters (A) — The A in a character attribute position allows the operator to input the letters A through Z (either in uppercase or lowercase), 8-bit characters with octal values from 300 through 375, and a space.

2.6.7.3 Unsigned Numbers (9) — The 9 in a character attribute position allows the operator to input only the digits 0 through 9.

2.6.7.4 Signed Numbers (N) — The N in a character attribute position allows the operator to input the digits 0 through 9, with only one decimal point and with only one plus (+) sign or one minus (−) sign. Their positions within the field are not checked by the Form Driver. Any other input is rejected.

2.6.7.5 Any Printable Characters (X) — The X in a character attribute position allows the operator to input any displayable character.

2.6.7.6 Mixed Pictures — A single field can contain different picture-validation characters. For example, a field constructed to accept both alphabetic and numeric characters might look like this:

```
AAA999
```

Such a field allows the operator to enter alphabetic characters in the first three field character positions and digits in the last three field character positions. The field is said to have a "mixed picture."

2.6.7.7 With Field-Marker Characters — The Form Editor treats all field-marker characters — whether leading, trailing, or embedded — as part of the field in which they occur. See Table 2-3 for a list of field-marker characters.

For example, a field that contains two field-marker characters, the pound sign and the dash, looks like this:

```
999#AA-99
```

A field that contains field-marker characters but only one picture-validation character does not have a mixed picture. Two or more picture-validation characters in a single field constitute a mixed picture.

A field can contain the ASCII characters from 41 to 57 octal and 72 to 100 octal as field-marker characters (Table 2-3). The Form Editor accepts field-marker characters when in FIELD mode. The Form Driver does not return field-marker characters to the calling task or include them in the length of the field. Field-marker characters are transparent to the task, which does not pass them to the Form Driver in the data to be displayed in a field.

Table 2-3. Field-Marker Characters

Character	Character	Character
!	(	:
"	)	;
#	*	<
$	+	=
%	,	>
&	-	?
'	.	@
/		

2.6.8 Assigning Form-Wide Attributes

The Form Editor collects form-wide attributes by displaying the questionnaire shown in Figure 2-3. The Form Editor automatically displays the Form-Wide Attributes Questionnaire when you create a new form or when you type FORM in response to the `COMMAND:` prompt. The questionnaire contains the default conditions for each choice.

```
Form Name                           :
Help Form Name                      :
Reverse Screen (Y,N)                N
Current Screen (Y,N)                N
Wide Screen                         (Y,N) N
Starting Line                       (1,23) 1
Ending Line                         (1,23) 23

Impure Area                         bytes
Form Size                           words
```

Figure 2-3. Form-Wide Attributes Questionnaire

Press the TAB key to move from one question to the next. When you have completed the necessary input and want to exit, return to the `COMMAND:` prompt by pressing the ENTER key.

The following sections define the fields listed in the Form-Wide Attributes Questionnaire.

2.6.8.1 Form Name — A response to this field is required. The name of the form is used in the form library directory and by Form Driver calls.

A form is always saved in a file with the name *form*.FRM. When inserted in a library, the name of the form is always taken from the form itself. The form name cannot contain embedded blanks.

2.6.8.2 HELP Form Name — This field contains the name of an associated HELP form. You can leave this field blank.

2.6.8.3 Reverse Screen — If this field contains a Y, the form is displayed black-on-white. If it contains an N, the display is white-on-black. The default is N.

2.6.8.4 Current Screen — If this field contains a Y, the Form Driver displays the form in the current screen mode. An 80-column form with a Y answer to this field does not require a change if the current mode is set at 132 columns.

The current screen also applies to reverse screen. If current screen is specified, the Form Driver does not change the screen background or the screen width, unless the form is specified for 132 columns and the screen is currently 80 columns.

You cannot specify Y both to this option and to the wide screen option described below. If the choice is N, the Form Driver resets the screen if necessary to conform to the display mode for this form. The default value is N.

2.6.8.5 Wide Screen — If this field contains a Y, the form is displayed in 132-column mode and the Current Screen option described above is set to N. If the field contains an N, the form is displayed in 80-column mode. The Form Editor changes the terminal to the selected mode. The default is N.

2.6.8.6 Starting Line — This field contains a value from 1 to 23 inclusive, indicating the first line of the screen to be cleared when the form is displayed. If you specify a starting line number greater than the ending line number, the Form Editor automatically replaces your entry with the default value of 1.

2.6.8.7 Ending Line — This field contains a value from 1 to 23 inclusive, indicating the last line of the screen to be cleared when the form is displayed. If the value is less than or equal to the starting line number, the default value of 23 is used.

Starting and ending line number defines the area of the screen to be cleared when the form is displayed using the FSHOW call (which does not automatically clear the entire screen) or when the form is displayed as a HELP form.

2.6.8.8 Impure Area — This is a display-only field that indicates the length of the impure area required when the form is displayed by the Form Driver. For applications written in a high-level language, the impure area must be 64 bytes larger than specified by FED. When you create a new form, this field is initially displayed as question marks.

2.6.8.9 Form Size — This is a display-only field that indicates the length of the form. This value is used in calculating the media or memory storage requirements for the form. When you create a new form, the field is initially displayed as question marks until the Form Editor determines the correct value.

2.6.9 Assigning Field Attributes

The Form Editor collects field attributes by displaying the Field Attributes Questionnaire shown in Figure 2-4. Each entry in the questionnaire designates a single attribute for a field. If the form or field is a new one, the Form Editor supplies default values in the questionnaire. If the attributes for the field were assigned in a previous editing session, those values are displayed.

```
Name       :  Right Just (Y,N) N Clear Char (chr) Zero Fill (Y,N) N
Default    :
Help       :
Autotab (Y,N) N Resp Reqd (Y,N) N Must Fill (Y,N) N Fixed Dec (Y,N) N
Indexed (N,H,V) N Disp Only (Y,N) N Echo Off (Y,N) N Supv Only (Y,N) N
```

Figure 2-4. Field Attributes Questionnaire

Enter the attribute assignment phase by typing ASSIGN and any of the following options in response to the `COMMAND:` prompt.

ASSIGN NEW	Assign attributes only to new fields.
ASSIGN ALL	Assign or edit attributes for all fields in the form.
ASSIGN FIELD *fldnam*	Assign or edit attributes for the field named *fldnam*.

ASSIGN is used to assign field attributes after creating the form's screen image. Any new fields placed in the form may have their field attributes defined by using either ASSIGN or ASSIGN NEW. If you exit from the field attribute assignment phase and then return to change any assigned fields, you must use the ASSIGN ALL or ASSIGN FIELD commands.

Fields that have changed their locations as a result of the OPENLINE or DELLINE operations, or as a result of character insertion or deletion on another part of the line, are recognized as existing fields. Fields whose pictures are modified must be redefined.

If you select ASSIGN NEW or ASSIGN ALL, you proceed to assign attributes to the next field by pressing the ENTER key. To return to the `COMMAND:` prompt before you have finished all the fields, type the period (.) on the keypad. (Remember that this results in assignment of default values to all remaining fields in the form. The default value for a field name is 6 blanks.) If you used ASSIGN FIELD, the Form Editor returns to the `COMMAND:` prompt when you press ENTER to terminate attribute assignment for that field.

The Form Editor displays the Field Attribute Questionnaire for each field on the form. The TAB key moves the cursor from one field to the next within the questionnaire. Pressing the ENTER key after going through the questionnaire for the last field in a form causes the Form Editor to reissue the `COMMAND:` prompt.

NOTE

> The assignment of invalid combinations of attributes to a field will result in an error message. To continue, press the ENTER key to redisplay the questionnaire for that field and correct the attribute that caused the error message.

The following field attributes appear in the Field Attributes Questionnaire:

Name	The name by which the field is known and referred to by your task. Unique field names are not required if a form is to be accessed by the FGETAL call. However, if the application is to access one field at a time, unique names should be assigned. (If a form contains more than one field with the same name, the Form Driver can access only the first one.) The default value for a field name is 6 blanks.
Right Just	If you type a Y, the field is right-justified. If you type an N, the field is left-justified. A Right Justified field may not contain a mixed picture. The default is N.
Clear Char	The character that you type in this field is displayed in place of the fill character (either zero or blank for the field). For zero-filled fields, it must be a zero. For blank-filled fields, the clear character may be any character. Underline, period, and blank are the most common choices. A blank is the default.
Zero Fill	If you type a Y, the field is filled with zeros before the operator enters any data in the field. If you type an N, blanks are stored in the field. Note that the Clear Character attribute must be set to zero if the field is zero-filled. The default is N. The fill character is also returned to the calling program in any positions where the terminal operator does not enter data.

Default	Specifies the initial value to be stored in the field when the form is loaded by the Form Driver. If you do not respond, the field contains either blanks or zeros, depending on your response to the Zero Fill attribute. Your answer to Default should be consistent with the picture-validation type of the field. If a default value is not specified, the internal representation of the field is blank or zero-filled depending on the definition. The fill character is always displayed as the clear character. If a field has no default value, it is initially displayed with clear characters. The default value must not be longer than the field.

NOTE

The Form Editor does not validate default data values to be certain that they are legal and conform to the picture-validation type of the field.

HELP	Specifies a line of information associated with the field that the user can read by pressing the HELP key. The HELP message appears on the last line of the terminal screen. The default is that no HELP message is displayed. If this field is left blank, the HELP form for the entire form is displayed if there is one. Otherwise, the message "NO HELP AVAILABLE" is displayed.
Autotab	Determines whether entering the last character in the field causes the cursor to advance automatically to the next field. Typing a Y specifies that Autotab is in effect. The default is N.
Resp Reqd	At least one character that is not the fill character must be entered in the field. If you type a Y, the operator at the terminal must respond to the field with some kind of input before continuing. If you type an N, the operator does not have to respond to the field. The Form Driver uses this attribute to validate the operator's responses for fields. The default value is N.
Must Fill	If data is entered in the field, the field must be filled so that it does not contain a single fill character. The field must be either empty or full. The default value is N. A field defined as Must Fill, but not Response Reqd, must be filled by the operator only if he or she enters data in it. It can be left empty.

Fixed Dec	If you type a Y, the field is a fixed-decimal field, provided that the picture is all 9s with an embedded decimal point. Signed numeric is not valid. If you type an N, or if the numeric picture-validation type is not in effect, the field is not fixed decimal. The default value is N.
Indexed	This attribute enables you to define identical fields, one below the other, as indexed fields. Typing an N indicates that the field is not indexed. Typing an H indicates that the field is horizontally indexed and that the cursor should proceed horizontally to the next indexed field on the same line in response to the TAB key (or Autotab) if the next field is also horizontally indexed. Typing a V indicates that the field is vertically indexed and that the cursor should proceed vertically to the next field in the same column in response to the TAB key (or Autotab). The default value is N. The Indexed attribute is illegal for fields in scrolled areas.
Disp Only	If you type a Y, only your application program can place data in the field. If you type an N, both the terminal operator and your program may enter data in the field. The default is N.
Echo Off	If you type a Y, data in the field is not displayed on the terminal screen. If you type an N, the characters echo as in normal operation. The default is N.
Supv Only	If you type a Y, the field is display-only, unless the program has turned off supervisor-only mode (by means of a Form Driver call). If you type an N, the field is not display-only and can be accessed by the terminal operator. The default is N.

The attribute that defines a field as scrolled does not appear on this questionnaire. You can define scrolled lines by pressing the GOLD/SCROLL key sequence during the EDIT phase.

2.6.10 Assigning Named Data Attributes

Named data is any data that is to be associated with a form but not displayed with it. Usually, named data contains information that the application uses to control task flow in a form-dependent manner. The information can consist of the names of other forms or task modules. Named data might also contain field specific data. Your task accesses named data with calls to the Form driver.

The Named Data Questionnaire in Figure 2-5 collects named data consisting of two horizontally-indexed fields of 16 elements. When the questionnaire appears, it includes all existing named data followed by blank named data fields.

Enter the named data phase by typing NAME in response to the `COMMAND:` prompt. The Form Editor displays the Named Data Questionnaire. When the questionnaire appears on the screen, the cursor is at the first character position of the first field. Enter the name by which you want to reference the data that you supply next. After you enter the name, press the TAB key to move into the data field. Now enter or edit the named data itself. If a name already exists, simply TAB over to the data field. Exit from the named data phase by pressing the ENTER key. The fields in the Named Data Questionnaire (Figure 2-5) are:

NAME	A 6-character field that receives the name of the data item. Form Driver calls access a particular element of named data by using either its name or its index number in the list of named data for a form.
DATA	A 60-character field that receives the data.

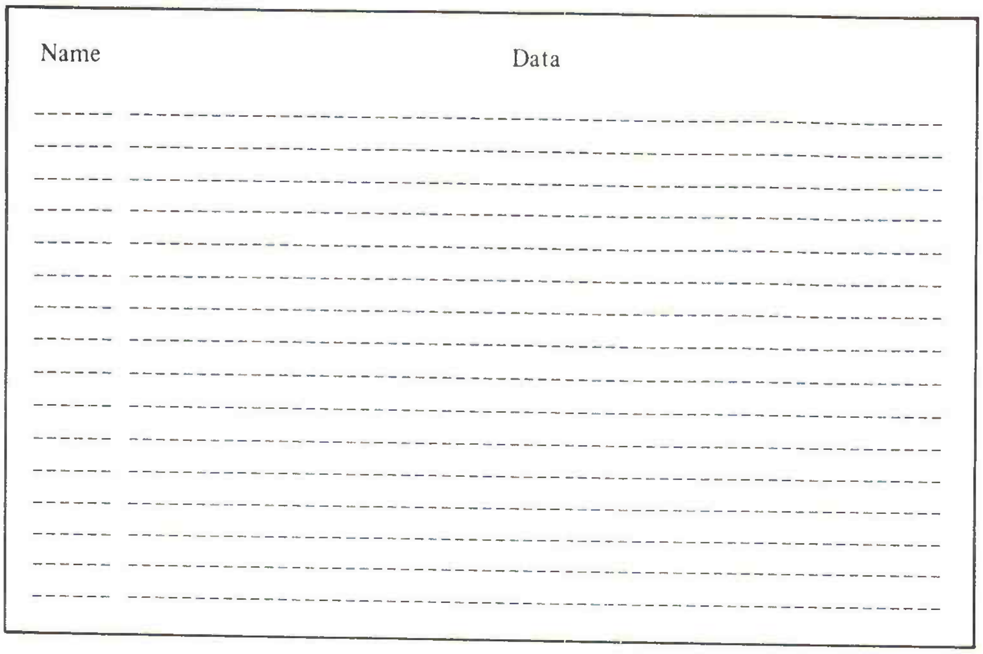


Figure 2-5. Named Data Questionnaire

2.7 How to Use the Form Editor

This section presents a step-by-step example of creating and modifying screen versions of forms. This example demonstrates some of the most common Form Editor commands, functions, and design processes. However, the example does not cover all Form Editor features, and it is not a complete tutorial. The purposes of this example are as follows:

- To illustrate how you can design a computerized version of a simple printed form.
- To show you what the screen looks like while you are working with the Form Editor.
- To introduce how the Form Editor uses the VT200 special function keypad to control editing functions.
- To introduce how the Form Editor uses the Form Driver and special questionnaires that collect information from you about the form you are designing.

This example has three major stages. The first stage describes a printed form. Assume that the form was originally designed for a card file of a company's vendors. Before designing the computerized version of the form, read the requirements of the fields in the original form (see Section 2.7.1.2).

In the second stage, you create the screen version of the form. Each step in this stage starts with an instruction, and each step completes a part of the exercise of designing the computerized version of the sample form. Read the instruction, and then look at your screen while you follow the instruction. Watch how the Form Editor responds. Finally, read the explanation that follows the instruction.

In the third stage, you modify one of the demonstration forms that you received as part of your FMS software kit. Use the same procedure for the steps in this stage as for the second stage.

You are encouraged to try using this example. You will be able to add the new form that you create and the demonstration form that you modify to the demonstration form library file, DEMLIB.FLB. You can then demonstrate how your new form works by running one of the demonstration programs supplied in your FMS software kit.

2.7.1 The Printed Form

The first steps in designing a screen version of the form are:

1. Provide an overview or a rough draft of the new form.
2. Describe the requirements for each field.
3. Describe the layout of the form and any special video features that it is to include.
4. Sketch the screen form and include the maximum lengths of fields.

2.7.1.1 Overview of the New Form — The new form will have fields for all of the information the printed form can contain. This example assumes that the new form will be used only to enter the vendor information currently in a card file.

VENDOR is the name to be assigned to the form. For now the form will not have a HELP form associated with it. It will use the 80-column screen width and the full screen height (Screen Lines 1 through 23).

2.7.1.2 Requirements of the Fields in the Original Form — This section describes all the requirements for the fields in the original form.

1. Vendor Number

   Vendor numbers are in the following form:

   B-67-0085

   The first character can be any letter. Except for two hyphens as shown, the remaining characters must be digits. An operator must enter the vendor number. Programs that use the form can then use the vendor number to get other vendor information from a computer file and display that information.

2. Vendor Name

   Vendor names may be as long as 38 characters and may include any printable character. When you first enter the name, you must type it exactly as it appears on the file card.

3. Address

   The top line in the address shows the vendor's street address. The next line shows the city and state. The bottom line shows foreign countries and mail codes, such as the Zip Code.

4. Contact

   Contact names are the names of the people in the vendor companies who are most informed about the sample company's business. Contact names can be as long as 28 characters and can include any printable character.

5. Phone

   The form is designed for one standard North American telephone number in the following format:

   (123) 555-4678

   As shown, parentheses enclose a 3-digit area code. A 7-digit number has a hyphen separating the exchange code from the line number. All input characters must be numbers. The telephone number is not required information, but if a telephone number is entered, all 10 digits must be entered. The area code has the default value 111 because most vendors are in that area, but there is no default value for the balance of the telephone number.

6. Extension

   The form must be designed for two telephone extension numbers. To make the new form as flexible as the original printed form, the new form will accept extension numbers up to seven digits, to cover the cases when different vendor extensions are complete 7-digit telephone numbers. The telephone extension is not required information. All input characters must be numbers, but any number of characters is valid.

2.7.1.3 Layout and Video Features of the New Form — The layout of the new form will follow the sketch that appears in Figure 2-6. With a screen width of 80 columns, there is ample room. Abbreviations are not necessary. The example assumes that the vendor number is the most important piece of vendor data, and therefore, the sketch shows it at the upper-left corner of the form.

The Vendor Number field is in reverse video and underlined to show its importance. Other fields will have the bold video attribute to make the values that operators enter more visible. The title of the form, "Vendor Data," also has the bold video attribute. Field labels will be in standard video. (The bold video attribute really doesn't look good in a form if it's used as much as specified here.)

2.7.1.4 Sketch of the Form Named VENDOR — Figure 2-6 is a sketch of the form that you will be creating in this example. Several other designs would be equally effective. In many cases the sketch you use can be less detailed than the one in Figure 2-6. Since you can easily change any design by using the Form Editor, you need only enough detail in a sketch to show the number of fields on each line and the rough alignment of fields. More detail appears in Figure 2-6 to increase the reliability of this sample.

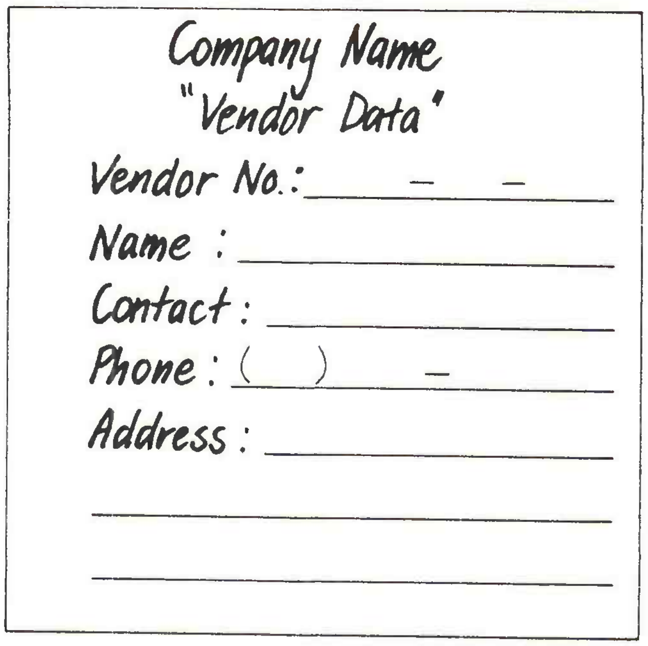

Company Name
"Vendor Data"
Vendor No.:_____ - _____ - _____
Name : ____________________
Contact : ____________________
Phone : (___) _____ - _____
Address : ____________________

Figure 2-6. Sketch of VENDOR

2.7.2 Creating the Screen Form (Steps 1. and 2.)

This section guides you from the start of the Form Editor through each of the other steps you must complete to create a screen version of the form named VENDOR. Each step starts with an instruction. Read the instruction, and then look at your screen while you follow the instruction. Watch how the Form Editor responds. Finally, if an explanation follows the instruction, read it and then go on to the next step.

The first time you work with this example, follow each instruction carefully. Each step has been written to depend closely on the preceding step.

1. Log in to a system that includes the Form Editor. (Check with your system manager if you are not sure whether the Form Editor is available on your system.)
2. To set up your VT100 or VT200 terminal, consult the appropriate terminal user guide. As a general rule, you will need to set up your terminal for block cursor, 80-column screen width, standard video display (light characters on a dark background), and signaling with the terminal bell. Your user guide has detailed instructions for terminal setup.

NOTE

> If your system runs with Packetnet System Interface (PSI) and you will be using X.25 with FMS, you must first adjust the CCITT X.3 parameters or FMS will not operate with your system's packet assembly/disassembly (PAD). Proper adjustment for your terminal entails disabling the Data Forward and T1 Idle Timer facilities, to allow all characters to pass forward immediately on data entry. Before you attempt to run FMS with X.25, be sure to talk to your packet-switching vendor to determine what values should be assigned to these parameters.

2.7.2.1 Starting the Form Editor (Step 3.)

3. Start the Form Editor by entering the following command or sequence of commands from the following list. The prompts that your system types are in black. The responses that you should type are in red.

   For RSX-11 and 11M-PLUS systems

   ```
   MCR>RUN $FED  RET
   FED>/CR  RET
   ```

   Explanation:
   These commands start the Form Editor and allow screen form development to begin. The Form Editor responds by displaying the Form-Wide Attributes Questionnaire.

2.7.2.2 Assigning the Form Name (Steps 4. and 5.)

4. Enter the name VENDOR. If you make a mistake, press the DELETE key to erase incorrect characters, and then complete the form name correctly. Later steps depend on the fact that the name of the new form is VENDOR. When the form name is correct, press the RETURN key.

   Explanation:
   Each character you type appears in the Form Name field. The cursor advances through the field from left to right.

   When a questionnaire is displayed, press the RETURN key to do the following:

   - Assign to each questionnaire whatever value you put into the field.
   - Store the questionnaire information internally until you change it or save the form description you are creating.

- Erase the questionnaire from the screen and in some cases ask for a Form Editor command.

- Continue a process by changing your display in some other way.

With the Form-Wide Attributes Questionnaire displayed, the RETURN key always causes the Form Editor to erase the screen and prompt for a Form Editor command by displaying the prompt `COMMAND:` on the last line.

5. Type the command EDIT. Press the DELETE key to correct mistakes. When you complete the command, use the ENTER function — press the ENTER key on the keypad, or press the RETURN key.

   Explanation:
   The ENTER or RETURN function keys cause the Form Editor to execute the command you have just typed. When the Form Editor executes the EDIT command, it displays the screen form that you are designing and shows you each detail of the form that you have specified so far. In this case, your new form is entirely blank - 23 lines long, with 80 spaces in each line. The cursor appears in the upper-left corner of the screen on Line 1 and Column 1.

   While you are editing a form, the Form Editor uses Line 24 to show you information about the cursor's location and several Form Editor settings that you change while you are editing. At this point, the different sections of Line 24 and their meanings are:

   - `CURSOR TXT NOR LIN 1 COL 1`

     The cursor is on a text character (`TXT`) and the line is a normal line of a form (`NOR`), not a scrolled line. (Scrolling features are explained later in this chapter.) The cursor's position is on Line 1 and Column 1.

   - `MODES TXT OVS ADV`

     The description of current settings of the editing modes; later steps demonstrate the effects of the different modes.

     – The TEXT mode (`TXT`) is for entering background text. The FIELD mode allows you to assign attributes to each field label you created in TEXT mode.

     – The OVERSTRIKE mode (`OVS`) is for replacing the character that the cursor is on with the characters that you type.

     – The ADVANCE mode (`ADV`) is for advancing the cursor to the right and down when certain cursor movement functions are used.

   NOTE

   While using the Form Editor, you will be using the Form Editor auxiliary keypad as well as the main keyboard to perform specific form editing functions. Refer to the Form Editor keypad layout in Figure 2-2.

2.7.2.3 Creating the Background Text (Steps 6. through 20 .)

6. Use the DOWNARROW function to move the cursor to Line 2 by pressing the DOWNARROW key once.

   Explanation:
   The DOWNARROW function moves the cursor straight down one line at a time. The Form Editor reports the cursor's new position in Line 24.

7. With the cursor on Line 2 and Column 1, type the name of your company or any other company name that you would like to use. Press the DELETE key to correct mistakes.

8. Use the LEFTARROW function (by pressing the LEFTARROW key) to move the cursor back to Line 2 and Column 1. Press the key several times.

9. Use the INSERT function to set the Form Editor to insert mode. Press the 9 key on the keypad.

   Explanation:
   The standard function of the 9 key on the keypad is the INSERT function. The function sets the Form Editor to insert mode. The abbreviation `INS` replaces `OVS` in the modes section of Line 24. In the INSERT mode, the Form Editor moves characters out of the way of insertions rather than replacing the characters.

10. Move the company name to the right in Line 2 by inserting spaces at the beginning of the line. Insert spaces until the company name is centered in Line 2 on Column 39 or 40. Hold the space bar down for each space you want to insert.

11. Use the BLINE function to move the cursor to Line 3 and Column 1. Press the 0 key on the keypad.

    The standard function of the 0 key on the keypad is BLINE. In the advance mode, the BLINE function advances the cursor to the next line and Column 1. In the BACKUP mode, the BLINE function backs up the cursor to Column 1.

12. Use the following sequence of functions and keyboard keys to move the cursor to Column 34.

    Press the PF1 key on the keypad, then type 33 on the keyboard, and finally press the RIGHTARROW key.

    GOLD 33 RIGHTARROW

    Explanation:
    The PF1 key is used only for the GOLD function. When the operator uses the GOLD function before typing a number on the keyboard and then uses another Form Editor function, the Form Editor repeats the last function as many times as specified. In this case, the Form Editor repeats the RIGHT-ARROW function 33 times and the cursor moves from Column 1 to Column 34.

13. Type the title of the form, Vendor Data, or any other title you would like to use. With the cursor at Column 34, the title Vendor Data will be centered.

14. Move the cursor back to Line 2. Press the BACKUP key on the editor keypad, then press the BLINE key to get to Line 2. Press the DELCHAR key to remove spaces and to center the company name.

15. Move the cursor to Line 5 and Column 1, using the ADVANCE and BLINE functions.

16. With the cursor on Line 5 and Column 1, type the field label for the Vendor Number field, `Vendor Number:`.

17. Use the RIGHTARROW function to move the cursor to Column 34. Press the RIGHTARROW key and watch the column number in Line 24. You can also press GOLD 19 RIGHTARROW. Then type the label for the Vendor Name field, `Name:`.

18. Use the DOWNARROW and LEFTARROW functions to move the cursor to Line 6 and Column 34, directly under the N of Name in Line 5. Then type the label for the vendor contact field, `Contact:`.

19. Use the DOWNARROW and LEFTARROW functions again to move the cursor to Line 7 and Column 34. Then type the label for the Vendor Telephone field, `Phone:`.

20. Use the DOWNARROW and LEFTARROW functions again to move the cursor down two lines to Line 9 and Column 34. Then type the label for the three Vendor Address fields, `Address:`.

2.7.2.4 Creating the Fields (Steps 21. through 29.)

21. Step 21. through 29. create the fields whose labels you have typed. Use the BACKUP function to change the directional mode and the BLINE function to move the cursor back to Line 5 and Column 1. Press the 5 key on the keypad once and the 0 key on the keypad several times until the cursor is back on Line 5.

    Explanation:
    The standard function of the 5 key on the keypad is the BACKUP function. The BACKUP function sets the Form Editor to BACKUP mode. The abbreviation BCK replaces ADV in the modes section of Line 24. In BACKUP mode, the BLINE function backs up the cursor directly to Column 1.

22. Move the cursor to Line 5 and Column 15. Use the OVERSTRIKE and FIELD functions to set the Form Editor to the OVERSTRIKE and FIELD modes. Press the following sequences of keys:

    - For the OVERSTRIKE function, the PF1 key and then the 9 key on the keypad.
    - For the FIELD function, the PF1 key and then the 8 key on the keypad.

    Explanation:
    The alternate function of each keypad key is the function whose name is at the bottom of the key in the keypad diagram. The alternate function of the 9 key on the keypad is the OVERSTRIKE function, and the alternate function of the 8 key on the keypad is the FIELD function. To use an alternate function, press the GOLD key first and then press the key that controls the function you want to use.

    The OVERSTRIKE function sets the Form Editor to overstrike mode, as described earlier.

    The FIELD function sets the Form Editor to FIELD mode. To create a field, the Form Editor must be in FIELD mode. In FIELD mode you can type only field picture characters and field format characters.

    The full sets of field picture and field format characters are described later in this chapter. In this example you will use only the field characters listed in Table 2-4.

23. In this example, vendor numbers are in the following form:

    B-67-0085

    To create the field for the vendor number, type `A-99-9999`.

    Explanation:
    `A-99-9999` specifies the valid characters for each column in the field; these make up a field picture. The picture specifies that the first character in the field must be a letter or a space and the other characters must be digits. The hyphens separate parts of the field. For a program that processes the field the hyphens will not be part of the field value. Therefore, the program uses only seven characters, although nine characters display.

24. Move the cursor to Line 5 and Column 43. Create the Vendor Name field by inserting the letter X 37 times. The easiest way to do this accurately is with the following sequence:

    GOLD `37` X

    Press the PF1 key, type 37 on the keyboard, and press the X key.

Explanation:
The GOLD function sequence for repeating functions also repeats characters you want to insert.

Any character can appear in a vendor name. Therefore, the form has to allow any character.

25. Move the cursor to Line 6 and Column 43. Create the Vendor Contact field by inserting the letter A 28 times. Use the following sequence:

    GOLD 28 A

    Explanation:
    Assume that only spaces and letters can appear in the contact name. Periods (.) after initials and abbreviations will not be copied from the card file. If an operator types a period or other invalid character, the Form Driver will refuse to accept the character and will signal the operator with the following message:

    ```
    ALPHABETIC REQUIRED
    ```

26. Move the cursor to Line 7 and Column 43 by pressing GOLD 42 RIGHT-ARROW. Create the Vendor Phone field by typing `(999)999-9999`.

    Explanation:
    Assume that only the digits 0-9 can appear in a phone number. If old phone numbers that include letters in the exchange code are copied, the operator should convert the letters to the corresponding numbers.

27. Move the cursor to Line 9 and Column 43 by pressing the DOWNARROW key twice, and the LEFTARROW key until you see 43 in the Edit Status Display field column on the bottom of the screen. Create the first Vendor Address field by inserting the letter X 28 times.

    Explanation:
    Assume that any character can appear in an address.

28. To experiment with duplicating a field without retyping it, move the cursor back to the first X in the Vendor Address field. Then use the following sequence of functions to erase the field picture and restore it to the form description:

    GOLD DELEOL GOLD UNDELLINE

    Press the PF1 key, the 2 key on the keypad, the PF1 key again, and the PF4 key.

Explanation:
The alternate functions of the 2 key and the PF4 key are DELEOL and UNDELLINE. The DELEOL function erases the cursor's character and the other characters between the cursor and the end of the line. The Form Editor stores the erasure in an internal line buffer, in case you want to restore the last line you erased.

The UNDELLINE function restores the string in the line buffer to the form description. Therefore, when you want to create several fields with the same field picture, one easy method is to create a field picture, erase it, and then restore it in as many locations as needed.

29. Move the cursor to Lines 10 and 11. With the cursor in Column 43 in each line, create the other Vendor Address fields by using the UNDELLINE function.

2.7.2.5 Assigning Field Attributes (Steps 30. through 40.)

30. In Steps 30. through 40, you will complete the Field Attributes Questionnaire for each field you have created. To begin work with the Field Attributes Questionnaire, enter the ASSIGN command. Use the following sequence:

    GOLD COMMAND `ASSIGN` ENTER (or RETURN)

    Explanation:
    The alternate function of the 7 key on the keypad is the COMMAND function. After the COMMAND function, the Form Editor erases Line 24 and displays the prompt `COMMAND:`. When the prompt appears, enter a command by typing on the keyboard and use the ENTER function to cause the Form Editor to execute the command.

    The ASSIGN command causes the Form Editor to display the Form Attributes Questionnaire for each new field. A new field is a field for which no field attributes have been assigned. In this case, all the fields you have created are new. The first new field is the Vendor Number field. The Form Editor displays the Form Attributes Questionnaire so you can still see the field itself. Then the Form Editor identifies the field by replacing each picture character with an underline character (_). Within the Field Attributes Questionnaire, the cursor is displayed in the first field.

    Like the Form-Wide Attributes Questionnaire, the Field Attributes Questionnaire is also an FMS form displayed by the Form Driver. The full set of fields in the Field Attributes Questionnaire is explained later in this chapter. For this example, the field attributes you must assign are listed in Table 2-5.

31. For the Vendor Number field, type the field name NUMBER and press the TAB key to move to the next field in the questionnaire. Press the DELETE key to correct any typing errors.

    Explanation:
    When displaying a questionnaire, the Form Driver displays each character as you type it. The TAB key signals that you are finished with the Name field, although you can return to the field later and change it. The Form Driver responds by moving the cursor to the next field that you should complete. Table 2-6 lists the Form Driver editing functions that you will need in this example. The full set of editing functions is explained in Chapter 4.

32. Press the TAB key four times. With the cursor at the beginning of the Help field, type a short, helpful message that describes how an operator is to type a vendor number. For example:

    ```
    Copy the vendor number from the old vendor card.
    ```

    Press the DELETE or LINEFEED keys to remove mistakes.

    Explanation:
    Each time you press the TAB key, the cursor moves to the next field in the questionnaire. For the Right Just, Clear Char, and Zero Fill fields, the default field attributes are unchanged. Therefore, in your new form, the Vendor Number field will have the following corresponding attributes:

    - Not right-justified.
    - The space is the clear character. (It is better to assign a clear character such as underline or if space is used assign the reverse video attribute so the field is visible on the screen.)
    - Not filled with zeros.

    The Default field in the questionnaire remains blank. Therefore, in your new form, the Vendor Number field will not have a default value.

Table 2-4. Field Characters Required for the Example

Character	Usage
Field-Picture Characters	
9	For the positions in the vendor number and telephone number, where a number is the only valid character.
A	For the first position in the vendor number, where a letter is the only valid character.
X	For the vendor name, contact name, and vendor address fields, where any printable ASCII character is valid.
Field-Marker Characters	
(	For enclosing the area code in the telephone number.
)	For enclosing the area code in the telephone number.
-	For separating the two parts of the telephone number.

Table 2-5. Field Attributes Required for the Example

Attribute	Usage
Name	To provide a unique identifier for each field.
Default	To specify the most common area code that occurs in vendor telephone numbers.
HELP	To provide reminders to the operator about completing fields.
Response Required	To require the operator to enter the vendor number before finishing with the form.
Must Fill	To require the operator to enter all of the characters in the vendor number and telephone number.

Table 2-6. Form Driver Editing Functions Required for the Example

Function	Usage
BACKSPACE	To back up from field to field in a questionnaire.
DELETE	To erase a single character in a questionnaire response field.
LINEFEED	To erase an entire questionnaire response field.
RETURN	To signal that all responses are correct in a questionnaire.
TAB	To advance from field to field in a questionnaire.

33. When you have typed the HELP message, move the cursor to the Resp Reqd field, and type Y for "yes."

    Explanation:
    In your new form, the Vendor Number is required information. By typing Y, you assigned the Response Required field attribute. The Form Driver responds by moving the cursor to the next field (as though you had pressed the TAB key).

34. With the cursor on the Must Fill field, type Y.

    Explanation:
    In your new form, the operator response must fill the Vendor Number field. By typing Y, you assigned the Must Fill attribute. The Form Driver responds by moving the cursor to the next field automatically.

35. Press the RETURN key.

    Explanation:
    For the field attributes after the Must Fill field, the defaults are correct for the Vendor Number field. The RETURN or ENTER key signals that you are finished with the questionnaire. The Form Driver responds by displaying a fresh image of the Field Attributes Questionnaire. The Form Driver also identifies the next field in your new form, the Vendor Name field, as the field to which you should now assign field attributes. The cursor appears at the beginning of the Name field in the questionnaire.

36. Type VNAME as the field name. Move the cursor to the HELP field with the TAB key and type a HELP message such as:

    ```
    Copy the vendor's name from the old vendor card.
    ```

    Explanation:
    The other default attributes are correct for the Vendor Name field. Therefore, press the RETURN key when you complete the HELP message. The Form Driver identifies the next field in your new form, the Contact field, as the field to which you should now assign field attributes.

37. Type CONTAC as the field name. If you want to specify a HELP message for the Contact field, move the cursor to the Help field and type the message. The other default attributes are correct for the Contact field. Therefore, when the Name and Help fields are complete, press the RETURN key.

38. Type the name PHONE for the next field. Move the cursor to the default field and type 111 as the default area code. Then move the cursor to the Help field if you would like to assign a HELP message for the Phone field. One example of a HELP message is:

    ```
    The area code and a 7-digit number are required.
    ```

Explanation:
For the new form, the default area code is 111, although the design does not call for a default number. With 111 as the only printing characters in the default value, the field will look like the following example when the Form Driver displays your new form:

```
Phone: (111) -- -- --
```

39. Assign the Must Fill attribute to the Phone field. Advance the cursor to the Must Fill field and type Y. The other default field attributes are correct for the Phone field. Press the RETURN key when you have finished assigning the Must Fill attribute.

    Explanation:
    Although the telephone number is not required input data, if the operator types a number, all 10 columns of the area code and number must be complete. Therefore, the Must Fill field attribute must be assigned, but the Resp Reqd field attribute is not assigned. Since this field contains data already (the default value), it is going to have to be filled unless the default area code is deleted. The way Must Fill works is that if a field contains any data, it must be filled, as is the case here.

40. For each of the Address fields in your new form, complete the following procedure:

    - Assign field names to each - for example, ADDR1, ADDR2, and ADDR3.
    - Assign a HELP message, if you would like to do so.
    - For the other field attributes the defaults are correct. Press the RETURN key when you finish assigning the field attributes for each field.

    When you press the RETURN key after assigning the field attributes for the last Address field, you have finished assigning attributes to all fields in your new form. The Form Editor will return you automatically to the `COMMAND:` prompt.

2.7.2.6 Assigning Video Attributes (Steps 41. through 46.)

41. Assigning video attributes is part of the process of editing a form description. To illustrate the video attributes, the following steps guide you in making the following assignments:

    - Make the company name display in boldface.
    - Make the Vendor Number field label and field picture display in reverse video.

    With the `COMMAND:` prompt displayed in Line 24, type EDIT.

    Explanation:
    The Form Editor responds to the EDIT command by displaying your new screen form.

42. To assign video attributes to character positions in a form, you must first mark the positions by putting them in a select range using the SELECT function. Then you assign the combination of video attributes that you want to the select range.

    Move the cursor to the first character of your company name.

    With the cursor in that position, use the SELECT function. Press the period (.) on the keypad.

    Explanation:
    The Form Editor responds by adding information about your select range to Line 24. When you are building a select range, the Form Editor shows the line and column number of the cursor's position when you used the SELECT function.

43. Advance the cursor to the blank at the end of your company name. With the cursor in that position, use the VIDEO function. Press the 7 key on the keypad.

    Explanation:
    The Form Editor displays the `VIDEO:` prompt on Line 24.

44. To assign the bold video attribute to the select range, respond to the `VIDEO:` prompt by typing BOLD. Then press the ENTER key.

    Explanation:
    The Form Editor immediately displays the select range in boldface and again displays the `VIDEO:` prompt.

45. To finish assigning the video attributes, press the ENTER key without specifying a video attribute. Then advance the cursor to the V of the Vendor Number and begin to build a new select range by using the SELECT function.

    Explanation:
    The Form Editor updates the line and column numbers in the select range report in Line 24.

46. Advance the cursor to the blank that follows the field picture `(A-99-9999)` for the Vendor Number field, and use the VIDEO function. When the Form Editor displays the `VIDEO:` prompt, type REVERSE and press the ENTER key. To stop assigning graphic attributes, press the RETURN key.

    Explanation:
    The Form Editor responds by displaying the field label and picture in reverse video.

2.7.2.7 Assigning Named Data (Steps 47. through 50.)

47. This example assumes that you want to experiment with your new form by having the demonstration program display the form. To make that possible, you must assign named data to your new form. The demonstration program is listed and explained in Appendix C. The named data label that you need to assign is "NXTFRM" and the named data value to be associated with that label is the string ".NONE." Press GOLD COMMAND.

    With the `COMMAND:` prompt displayed on Line 24, type NAME and press the ENTER key.

    Explanation:
    The Form Editor responds by displaying the Named Data Questionnaire. The fields on the left in each line of named data are the fields for a label that are from 1 to 6 characters long. On the right is the data string that is from 0 to 66 characters long. The label is simply an identifier by which a program can request an associated data string (the Form Driver searches — not the program). The cursor is at the beginning of the first field in the questionnaire, the Name field.

48. To enter the label, type NXTFRM. To enter the data value, press the TAB key to move the cursor to the data field and then type .NONE.

    Explanation:
    As in the other questionnaires, the Form Driver is actually processing the entry form and your responses.

49. The form does not require any other named data. Therefore, press the RETURN key to get the `COMMAND:` prompt.

    Explanation:
    When you press the RETURN key while working with the Named Data questionnaire, the Form Editor displays the `COMMAND:` prompt.

50. You have now completed your new computerized version of the sample form. To save the form description that you have created, use the SAVE command. Complete the following sequence:

    `SAVE` ENTER

    Explanation:
    The Form Editor responds to the SAVE command by saving your new form description in an output file and displaying messages similar to the following:

    ```
    ?FED-Form being saved
    ```

    The Form Editor's prompt for a command line is also displayed.

2.7.2.8 Editing One of the Demonstration Forms (Steps 51. through 57.)

51. With the prompt displayed, you can continue to use the Form Editor to work on another form description or you can stop the Form Editor. The following steps assume that you want to have the demonstration program display your new form. To get the demonstration program to do that, you must modify the named data for the first form that the demonstration uses. The first form is a menu that is illustrated and explained in Appendix B. The form is named FIRST and is stored in the form library file DEMLIB.FLB. You need to modify the form as follows:

    - Add an alternative exercise to the list in the form by adding the following line of background text:

      ```
      4 Enter vendor data
      ```

    - Add a named data label and value to the other named data that are already associated with the form. The label and value are:

      ```
      5 VENDOR
      ```

    To edit the form named FIRST, respond to the `FED>` prompt by typing the following command:

    ```
    FED>DEMLIB.FLB (RET)
    ```

    When the Form Editor responds with the `Form name?` prompt, type FIRST and press the RETURN key.

    Explanation:
    The Form Editor displays the screen image of the form named FIRST and the `COMMAND:` prompt. The form has only one field, the single character field following the word DO. The field picture character 9 specifies that only numeric responses are valid for the field. All other characters in the form are background text.

52. Type EDIT. Then advance the cursor to the E in the line that reads `4 Exit`. With the cursor in that position, replace the word Exit with Enter vendor data. Check the report in Line 24 that you are in overstrike mode, and type the new phrase.

    Explanation:
    In overstrike mode, each character that you type replaces the character at the cursor's position.

53. To restore the choice of exiting from the demonstration program, insert the `Exit` choice. Advance the cursor to the character position directly below the `4` and type `5 Exit`.

54. The demonstration program uses the named data that are associated with the forms in DEMLIB.FLB to transfer control from form to form and to exit. Therefore, you must now change the named data that are associated with the form named FIRST so that:

    - The response `5` stops the demonstration.

- The response `4` makes the demonstration display the form named VENDOR and store vendor data in an output file.

To edit the named data associated with the form named FIRST, enter the NAME command. Use the following sequence:

GOLD COMMAND `NAME` ENTER

Explanation:
The Form Driver displays the Named Data Questionnaire, which has the data and labels that are associated with the form named FIRST. The cursor is at the beginning of the label in the Name field in the first line of the entry questionnaire.

55. Press the TAB key several times to advance the cursor to the label associated with `.EXIT` (4). Then, press the LINEFEED key to erase the 4, and type `5` to enter the new label.

    Explanation:
    When the Form Driver is displaying an entry form, press the LINEFEED key to erase all of the characters in any field. The cursor must be in the field you want to erase, but it can be at any character position in the field.

56. For programs that use named data labels to call for data, the named data associated with a form can be in any order. Therefore, to associate the response 4 with the form named VENDOR and the appropriate output file for vendor data, you can add the new named data at the end of the original data associated with the form. Do the following:

    - Press the TAB key until the cursor is in the first blank Name field of the entry form.
    - Type the label `4`, and press the TAB key to advance the cursor to the Data field.
    - Type `VENDOR`, the name of the form that is to be displayed for the response 4 to the form named FIRST. Press the TAB key to advance the cursor to the next blank Name field.
    - Type `4F`, a special label that the demonstration program will create. Press the TAB key to advance the cursor to the Data field.
    - Type `VENDOR.DAT` or another file name that you want the demonstration program to use for vendor data.
    - To finish editing the named data, press the RETURN key.

    Explanation:
    When you press the RETURN key while working with the Named Data Entry Form, the Form Editor displays the `COMMAND:` prompt.

57. To save the edited version of the form named FIRST, use the SAVE command. Type SAVE and press the ENTER key. The Form Editor saves a form description file named FIRST.FRM and displays a message similar to the one illustrated in Step 50.

2.7.2.9 Storing the New Forms in a Form Library File (Steps 58. through 59.)

58. The preceding step is the last one in this example that deals with the Form Driver and Form Editor. However, if you want to experiment with your new form and the edited version of the form named FIRST, you must add the form descriptions to the form library file DEMLIB.FLB. The FMS component that manipulates form descriptions and form library files is the Form Utility (FUT). The Form Utility is described in Chapter 3. The following steps provide the instructions that you need for the forms that you have just edited.

    With the Form Editor prompt displayed, stop the Form Editor by typing CTRL/Z. When the system prompt is displayed, start the Form Utility for RSX-11M and 11M-PLUS systems by using the following commands or sequences:

    ```
    MCR>RUN $FUT RET
    ```

    or

    ```
    FUT RET
    ```

59. With the Form Utility prompt (`FUT>`) displayed, type the following Form Utility command line:

    ```
    DEMLIB.FLB=DEMLIB.FLB,FIRST.FRM,VENDOR.FRM/RP  RET
    ```

    Explanation:
    The Form Utility produces a new version of the form library file DEMLIB.FLB. Because the command line causes the new version of the form named FIRST to replace the original version, the Form Utility reports the full file specification of the replacement. The report is a message that looks similar to the following:

    ```
    DP0:[220,10]FIRST.FRM;2 Form Name = FIRST
    Form replaced
    ```

    Therefore, the new version of the form library file DEMLIB.FLB now contains your edited copy of the form named FIRST and the new form description that you created for the sample vendor data form. You can use this version of DEMLIB.FLB instead of the distributed version when you run the demonstration program.

Chapter 3
The FMS-11 Form Utility (FUT)

The Form Utility (FUT) is the program that creates and modifies FMS-11 form libraries. You should use only the Form Utility when you want to examine FMS library files.

The Form Utility program provides the following services:

1. Extracts and deletes form descriptions from form libraries.
2. Combines form descriptions and form libraries into large form libraries.
3. Converts form descriptions to MACRO-11 object format for applications that use memory-resident forms.
4. Produce printable data descriptions in COBOL format and listing files for form library directories and form descriptions.
5. Creates form libraries from form files.

3.1 Starting and Stopping the Form Utility

You can run the Form Utility in two ways:

1. By calling the Form Utility directly with the task name FUT, if the Form Utility has been installed as a system utility.
2. By using the MCR command RUN.

3.1.1 Starting the Form Utility Directly with the Task Name FUT

With the Form Utility installed in your system, you can call the Form Utility directly by typing the task name. The two general formats of the call are:

1. `FUT file-specification-string[/options]` RET
2. `FUT` RET

In the first format, the types of the files in the file specification string and the options that you add to the string specify a Form Utility process. The Form Utility starts immediately and generally returns to the `MCR>` prompt when it finishes the process.

In the second format, the Form Utility first displays the prompt `FUT>`. When you respond to the prompt by typing a file specification string and options, the Form Utility starts the process you have specified and generally returns to the `FUT>` prompt when finished.

The different forms of the file specification string and the options that the Form Utility accepts are described in later sections of this chapter.

3.1.2 Using the RSX-11M/M-PLUS MCR RUN Command

On RSX-11M and RSX-11M-PLUS systems you can also start the Form Utility by using the MCR RUN command. The command and its options are described in the *RSX-11M/M-PLUS MCR Operations Manual.* With the Form Utility in the system account, the general form of the command is:

`RUN $FUT[/options]` RET

The Form Utility starts by displaying the `FUT>` prompt. Later sections describe how to respond to the prompt.

3.1.3 Stopping the Form Utility

The Form Utility stops in two ways, depending on how you start the Form Utility. The two general cases are:

1. When you start the Form Utility with the direct call FUT and include a Form Utility command line. In this case the Form Utility exits after completing the process you have specified and the system displays the `MCR>` prompt.

2. When you start the Form Utility with the RUN command or with the direct call `FUT` without a command line. In this case the Form Utility remains active after completing a process and displays the prompt `FUT>`. You can then enter a new file specification string or press CTRL/Z to exit.

3.2 Form Utility Defaults

Table 3-1 summarizes the command default values for the Form Utility for RSX-11M and RSX-11M-PLUS systems.

Table 3-1. Default Values for RSX-11M and RSX-11M-PLUS Systems

Item	Default
Input and output UIC	The LOGON UIC or the UIC specified in the latest SET/UIC command.
Input and output volume	The volume installed in the default device SY:.
Input file name and type	The input file name must be specified. The default input file type is .FLB.
Output file name and type	With the /FF option, no output file name or type can be specified. The form name becomes the file name and the file type is .FRM. With the /FD option, the form name is the default output file name. With the /LI option, the input form library file name is the default output file name. With the /CC option the output extension is .LIB. With other options, the output file name must be specified and the default file types are:
	.FLB for any output form library file.
	.FMD for printable form descriptions.
	.LIB for COBOL data descriptions.
	.LST for form library file directories.
	.OBJ for MACRO-11 object module memory-resident forms.
Input file version	The latest version of the input file version that is on the input volume.
Output file version	Version 1 for an entirely new file. Otherwise, the Form Utility assigns a version number that is one plus the version number of the latest version on the output volume.
Option	The default option is /FD, to produce a printable version of a form description.
Spooling and Block-Alignment	The /-SP and /BA options are the defaults for spooling and block-alignment of form descriptions. The default is to block-align the form library.

3.3 Form Utility Errors

When an error occurs, the Form Utility displays a message and transfers control in one of the three following ways:

1. When recovery is impossible, control transfers to your operating system.
2. When recoverable errors occur in processing form descriptions or files, control transfers to the `FUT>` prompt.
3. When file specifications and options control transfers to the `FUT>` prompt.

Appendix C lists Form Utility messages and explains how to look messages up.

3.4 Prompts for Form Library File Processes

The following six options allow you to select individual forms from form library files and process them in different ways:

1. /CC to produce a COBOL data description structure.
2. /DE to delete form descriptions from form library files.
3. /EX to select specific form descriptions from one form library file and store them in a new form library file.
4. /FD, the default option, to produce a printable form description.
5. /FF to extract a form description from a form library file and store it in a form description file.
6. /OB to convert form descriptions to object format for memory-resident purposes.

For each of the six options, the Form Utility prompts you for a form name. The general format of the prompt is the full file specification of the form library file followed by the prompt `Form name?`. For example, with DM0:, [30,10], and .FLB as the default input volume, UIC, and form library file type, and with Version 6 as the latest version of the form library file DEMLIB, the Form Utility would prompt you as follows:

```
MCR>FUT DESCR,FMD=DEMLIB/FD (RET)
DM0:[30,10]DEMLIB.FLB;6 Form name?
```

You can respond to the `Form name?` prompt by typing:

1. A valid form name and pressing the RETURN key.

The Form Utility processes only the form description for the form name that you type. It then requests another form name.

2. An asterisk (*) and pressing the RETURN key.

The Form Utility processes all form descriptions that the form library file contains.

NOTE

> Responding with an asterisk is not valid when you have specified the /FF option.

3. The RETURN key only.

   The Form Utility begins processing the next input file that you have specified, if there is another input file, or stops.

3.5 Form Utility Command Options

This section describes each of the Form Utility command options. The descriptions are arranged in three groups, as follows:

1. Options for control and HELP.
   - The /ID option to display the Form Utility identification.
   - The /HE option to display the Form Utility HELP file.
   - The /SP and /-SP options to control spooling of the files to the line printer.
   - The /LI option to list the names of forms in form library files.
2. Options for creating form library files.
   - The /BA and /-BA options to control form description block alignment.
   - The /CR option to create a form library file by combining files.
   - The /DE option to delete form descriptions from files.
   - The /EX option to extract form descriptions from files.
   - The /RP option to update form descriptions in files.
3. Options for processing and converting form descriptions.
   - The /CC option to create COBOL data declarations for form descriptions.
   - The /FD option to create a listing of a form description.
   - The /FF option to create a form description file from a form in a library file.
   - The /OB option to create MACRO-11 object modules of form descriptions for memory-resident forms.

3.5.1 Options for Control and HELP

3.5.1.1 The /ID Option: Displaying the Form Utility Identification — Use the /ID option alone in the command line to make the Form Utility display its identification. The identification includes the Form Utility's task name (FUT), version number, and patch level.

The following examples illustrate how the Form Utility responds to the /ID option:

```
MCR>FUT /ID  RET
FUT V02.00
MCR>

MCR>RUN $FUT  RET
FUT>/ID  RET
FUT V02.00
FUT>
```

3.5.1.2 The /HE Option: Using Form Utility HELP File — Use the /HE option alone in the command line to have the Form Utility display a short summary of the Form Utility command line forms, as well as a list of the command line options and their meanings.

Figure 3-1 shows how to use the /HE option and includes a copy of the Form Utility HELP display. Later sections in this chapter explain the other options fully.

```
RUN $FUT  RET
FUT>/HE  RET
                HELP FOR FUT V02.00

Command line:

          output-file = input-file, ... ,input-file/option

Options:

/ID       Print identification on terminal
/HE       Print this help text on terminal
/FD       Write form description (default)
/LI       Write library directory listing
/OB       Write object module of forms
/CR       Create library from libraries and forms
/RP       Replace forms in library
/DE       Delete forms from library
/EX       Extract forms to build library
/CC       Create COBOL form description
/FF       Create a form file from a library form
/-BA      Do not block align forms in library
/SP       Spool listing output to line printer

FUT>
```

Figure 3-1. The /HE Option and the HELP Display

3.5.1.3 The /SP and /-SP Options: Requesting Line Printer Listings — Use the /SP option with one of the following options to direct the Form Utility output to the default line printer on your system:

- With the /LI option, for form library file directories.
- With the /FD option, for printed descriptions of forms.
- With the /CC option, for COBOL data descriptions of forms.

When you use the /SP option, the Form Utility creates the output file and spools the file to your line printer after you specify the form name or indicate with an asterisk that you want all forms listed.

When you use the /-SP option with the same specifications, you prohibit line printer listings explicitly. The default option is /-SP.

If you don't specify output file with the /SP option, the default output device is the terminal, not the line printer.

3.5.1.4 The /LI Option: Listing Directories of Form Library Files — Use the /LI option to create a printable file that lists the names of the forms in form library files. The output file includes the following information:

1. The Form Utility identification and the current date.
2. The full file specification for the form library file.
3. The date the form library file was last updated.
4. The size of the directory within the form library file.

   This directory size is larger than one block only when the file contains an unusually large number of forms. To use a form library file whose directory is larger than one block in an FMS application, you will need to configure the Form Driver for the larger directory.
5. For each form description in the form library file:
   - The form name (as assigned by using the Form Editor).
   - The date the form description was last edited with the Form Editor.
   - The size of the impure area that the form requires in an application.

The following example illustrates the /LI option and the format of the output file that the Form Utility produces. Because the command line also includes the /SP option, the Form Utility spools the output file to the line printer after creating the file.

```
FUT   V02.00
4-JUN-85

Library    DMO:[30,10]DEMLIB.FLB;    created:    4-DEC-79
Directory    is    1    blocks    long

Form        Date        Impure Area (bytes)

FIRST     4-DEC-79          369
CUSTPR    4-DEC-79          326
LAST      4-DEC-79          275
EMPLOY    4-DEC-79          812
PARTS     4-DEC-79          794
CUSTO     4-DEC-79          612
```

3.5.2 Options for Creating Form Library Files

3.5.2.1 The /BA and /-BA Options: Using Block-Aligned Form Descriptions — Use the /BA option with one of the following options to align each form description explicitly from the beginning of a block to the output mass storage volume (/BA is the default option):

- With the /CR option.
- With the /DE option.
- With the /EX option.
- With the /RP option.

The input form library files can be aligned or unaligned.

Block-aligned form libraries might result in faster access times for an application. Block-aligned form descriptions require larger form library files than non-block-aligned form descriptions. The maximum increase in form library file size is one block for each form description.

In practice, block-aligned form library files are usually used unless space is severely limited. For example, if you are packaging an FMS-11 application with its form library files on RX01 diskettes or other media with small capacity, you might want to use non-block-aligned form libraries.

Use the /-BA option with the same specifications to prohibit block-aligned form descriptions.

Section 3.5.2.2 includes an example of the /BA option.

3.5.2.2 The /CR Option: Combining Form Library Files and Description Files — Use the /CR option to combine all input form descriptions in one form library file. The option has no effect on any of the input files. If the Form Utility finds a form name more than once in the input files that you specify, the following message is displayed:

```
FUT - Illegal replacement of form, use /RP
```

The following example illustrates use of the /CR option. Because the /BA option is also used, the form descriptions in the output file will be block-aligned.

```
MCR>FUT DEPT52.FLB/CR/BA=PROJ01,ECO12.FRM;5,PROJ23 (RET)
```

When the Form Utility completes the command, the form library file DEPT52.FLB contains the following form descriptions:

- All form descriptions in the latest version of the form library file PROJ01.FLB.
- The single form description in the form description file ECO12.FRM;5.
- All form descriptions in the latest version of the form library file PROJ23.FLB.

3.5.2.3 The /DE Option: Deleting Form Descriptions from Form Library Files — The /DE option lets you delete some form descriptions from form library files and combines the remaining form descriptions into a new form library file.

The Form Utility does not change any of the input files. For each input file that is a form library file, the Form Utility displays the full file specification and prompts you for the names of the forms you want to exclude from the output file.

NOTE

> The Form Utility accepts form description files as input files with the /DE option. However, none of the form description files will be combined in the output file. In effect, form description files are ignored in this case.

The following example illustrates the /DE option and the responses that exclude two forms from each of the input form library files:

```
MCR>FUT (RET)
FUT>FILMGD/DE=SLIDE.FLB;4,MOVIE.FLB;6 (RET)
DM0:[30,10]SLIDE.FLB;4 Form name?FIRST (RET)
DM0:[30,10]SLIDE.FLB;4 Form name?SECOND (RET)
DM0:[30,10]SLIDE.FLB;4 Form name? (RET)
DM0:[30,10]MOVIE.FLB;6 Form name?THIRD (RET)
DM0:[30,10]MOVIE.FLB;6 Form name?FOURTH (RET)
DM0:[30,10]MOVIE.FLB;6 Form name? (RET)
FUT>
```

When the Form Utility finishes, the form library file FILMGD.FLB;1 contains the following:

- All of the form descriptions that are in SLIDE.FLB;4 except for the forms named FIRST and SECOND.
- All of the form descriptions that are in MOVIE.FLB;6 except for the forms named THIRD and FOURTH.

3.5.2.4 The /EX Option: Extracting a Form Library File — Use the /EX option to extract form descriptions from form library files and combine them in a new form library file. When you include an input file that is a form description file, the Form Utility also adds the form description to the output file.

The Form Utility does not change any of the input files. For each input file that is a form library file, the Form Utility displays the full file specification and prompts you for the names of the forms you want to extract.

The following example illustrates the /EX option and the responses to extract two form descriptions from each input form library file:

```
MCR>FUT PICHLP/EX=SLIDE.FLB;4,005HLP.FRM,MOVIE.FLB;6 (RET)
DM0:[30,10]SLIDE.FLB;4 Form name?001HLP (RET)
DM0:[30,10]SLIDE.FLB;4 Form name?002HLP (RET)
DM0:[30,10]SLIDE.FLB;4 Form name? (RET)
DM0:[30,10]MOVIE.FLB;6 Form name?003HLP (RET)
DM0:[30,10]MOVIE.FLB;6 Form name?004HLP (RET)
DM0:[30,10]MOVIE.FLB;6 Form name? (RET)
MCR>
```

When the Form Utility finishes, the form library file PICHLP.FLB;1 contains the form descriptions for the forms named 001HLP, 002HLP, 003HLP, 004HLP, and 005HLP.

3.5.2.5 The /RP Option: Updating Form Descriptions in Form Library Files — You can use the /RP option to:

1. Replace a form description that is in a form library file with a new version that is in a form description file.
2. Create a form library file that contains all of the different forms (by form name) that are in several form library files.

The Form Utility can process the /RP option if each form stored in a form library file has a unique name. The Form Utility processes the input files one at a time from left to right. For input form descriptions with unique form names, the output form library file includes each one. For input form descriptions with the same form names, the output form library file includes only the last one processed. Therefore, the final contents of the output form library file in some cases depends on the order you used when typing input file specifications.

Figure 3-2 illustrates how the final contents of a form library file are different for two Form Utility commands. In the first case, the last version of the form named TEST02 that the Form Utility processes is in the input file TIN.FRM, and the output form library file includes only that version of TEST02. In the second case, the last version of TEST02 that the Form Utility processes is in the input file T.FLB, and the output form library file includes only that version.

Effects of Input File Order on Output File Contents

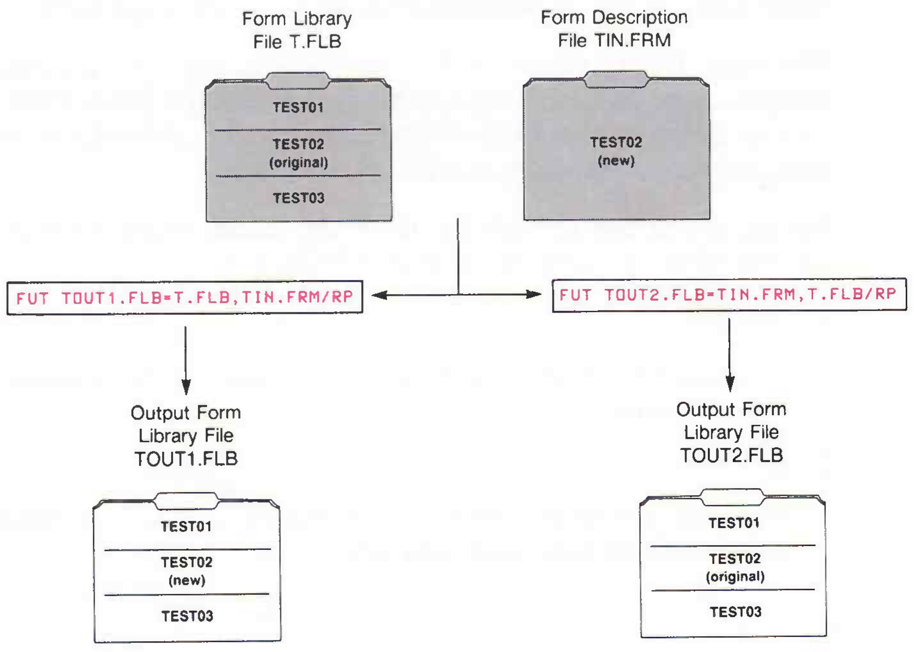

Form Library File T.FLB
TEST01
TEST02 (original)
TEST03
Form Description File TIN.FRM
TEST02 (new)
FUT TOUT1.FLB=T.FLB,TIN.FRM/RP
FUT TOUT2.FLB=TIN.FRM,T.FLB/RP
Output Form Library File TOUT1.FLB
TEST01
TEST02 (new)
TEST03
Output Form Library File TOUT2.FLB
TEST01
TEST02 (original)
TEST03

Figure 3-2. The /RP Option

3.5.3 Options for Processing and Converting Form Descriptions

3.5.3.1 The /FF Option: Creating Form Description Files from Form Library Files — Use the /FF option to extract a form description from a form library file and store the description in a form description file. With this option, you cannot specify an output file name and you cannot extract more than one form description at a time. To create a form description file from a form library file, use the general form:

```
form-library-file-spec/FF  RET
```

The Form Utility displays the full form library file specification and prompts you for the name of the form you want to extract. The Form Utility then uses the form name that you specify as an output file name, adding the file type .FRM.

In the following example, the /FF option extracts a form description from the form library file DM0:FRMLIB.FLB;5 in the account [100,30]. The Form Utility creates the form description file named HELP1.FRM.

```
>FUT  RET
FUT>SY:FRMLIB.FLB;5/FF  RET
DM0:[100,30]FRMLIB.FLB;5 Form name? HELP1  RET
FUT>
```

3.5.3.2 The /OB Option: Creating MACRO-11 Object Modules for Forms — Use the /OB option to convert form descriptions to MACRO-11 object format. You can then task build the object files with your FMS application to add memory-resident forms to the application.

The Form Utility processes the input files in the order that you type the file specifications. For each input form library file, the Form Utility prompts you for the names of the forms you want to convert. Conversion is automatic for each input form description file.

For each form that you specify, the Form Utility creates an object module with the following two program sections (PSECTs).

1. $FIDX$

   Contains the name of the form and a pointer to the beginning of the form's data structure.

2. $FORM$

   Contains the form description, including display attributes, default field values, named data, and field help.

The format used by the Form Utility for the object module is the same as for the following MACRO-11 module:

```
        .TITLE frmnam        ;Title of the module

        .IDENT /V02.00/      ;Form Utility version number
        .PSECT $FIDX$,D,GBL  ;Index that the Form Driver uses
        .RAD50 /frmnam/      ;to find the data structure for
                             ;a form that is called by name
        .WORD  FSTR          ;Pointer to the beginning of the
                             ;form data structure

        .PSECT $FORM$,D      ;Form data structure PSECT
FSTR:      .                 ;Form
           .                 ;      data
           .                 ;            structure
        .END
```

When you specify more than one form name, the Form Utility converts each form description and produces a concatenated object module of all the forms.

The following example illustrates the /OB option and responses to convert one form description from the form library file DM0:[30,10]BILL.FLB and the form description file DM0:[30,10]BILHLP.FRM to object format:

```
FUT BILFRM.OBJ=DR1:[30,10]BILL.FLB,DM0:BILHLP.FRM/OB RET
DM0:[30,10]BILL.FLB;6 Form name?  STARTUP  RET
DM0:[30,10]BILL.FLB;6 Form name?  RET
MCR>
```

The TKB command file order is important:

```
HLLCBL,FDVLRM/LB:FDVDAT,BILFRM.OBJ,FDVLRM/LB
```

The Form Driver data module (FDVDAT) must come before any memory-resident forms to be included in the task.

3.5.3.3 The /CC Option: Producing COBOL Data Declarations for Forms — Use the /CC option to produce an ASCII file that contains the data declaration statements that COBOL applications require for forms. You can then use the COBOL COPY statement in the data division of your COBOL program to refer to files that contain the data declaration statements, or you can use a text editor to add the data declaration statement file to your COBOL data division.

For each form that you specify, the Form Utility produces a three-level COBOL structure in the Terminal Format, as illustrated in Figure 3-3. COBOL also supports the Conventional (ANS) Format. You can use the COBOL REFORMAT utility to convert the Form Utility's data declaration structure to the Conventional Format. Refer to the *COBOL User's Guide* for a description of the REFORMAT utility and to the *COBOL Language Reference Manual* for a description of the Terminal and Conventional Formats.

Figure 3-3 shows a simple three-field form named PARTS and the COBOL data declaration structure that the Form Utility produces for the form. Assume that the field names PARTNO, DESCRP, and SUPPLR were assigned by using the Form Editor and that the Suppliers field has been designed as a vertically indexed field.

```
* COBOL Form Library Structure
*       This structure contains three types of data items:
*       Form Name, prefixed with "FORM-"
*       Named, prefixed with "N-", and
*       Data, prefixed with "D-".
01 FORM-PARTS-DEF.
  03 FORM-PARTS PIC X(6) VALUE "PARTS ".
    03 N-PARTS-PARTNO PIC X(6) VALUE "PARTNO".
    03 D-PARTS-PARTNO PIC X(9).
    03 N-PARTS-DESCRP PIC X(6) VALUE "DESCRP".
    03 D-PARTS-DESCRP PIC X(26).
    03 N-PARTS-SUPPLR PIC X(6) VALUE "SUPPLR".
    03 D-PARTS-SUPPLR PIC X(25) OCCURS 3 TIMES.
```

Figure 3-3. The /CC Option: Illustration of the COBOL Data Description

The following example illustrates the /CC option and responses to produce concatenated COBOL data descriptions for the form that is in the form description file DM0:[30,10]HELP04.FRM;2 and all forms in the form library file DM0:[30,10]FRMLIB.FLB;1.

```
MCR>FUT
D.LIB=DM0:[30,10]HELP04.FRM;2,DM0:[30,10]FRMLIB.FLB;1/CC  RET
DM0:[30,10]FRMLIB.FLB;1 Form name?*  RET
MCR>
```

NOTE

1. If the same name is used for more than one field in a form, the COBOL compiler will flag one of the fields as an error.
2. A COBOL data declaration cannot be created for a form description that contains blank field names.

3.5.3.4 The /FD Option: Producing Form Descriptions for Printed Listings — Use the /FD option to produce an ASCII file that describes all of the features of a form. You can print the file that the Form Utility produces or display it on your video terminal.

The printable description produced by the Form Utility is arranged in the following five major sections:

1. The form description header

   This section lists all form-wide information. For example, the form name, the associated HELP form name, and the impure area size the form requires.

   In unusual cases, the Form Utility might also detect a problem with a form that the Form Editor did not detect. In this case, the Form Utility prints a brief summary of each possible problem in this section.

2. The image map

   This section shows all of the constant text in the form and the default value that has been assigned to each field. When no default has been assigned, the image map shows the clear character that has been assigned.

3. The video attributes map

   This section shows the video attributes of all constant text and fields in the form.

4. The field descriptions

   For each field in the form, this section lists the field name, length, position, picture, clear character, and other assigned features.

5. The named data map

   This section includes a full list of the names, associated data, and order of the named data that have been assigned to the form.

   The following example illustrates use of the /FD option and responses to produce printable descriptions for one form from each of two form library files:

```
MCR>RUN $FUT RET
FUT>FD.TXT=DM0:[30,10]AM.FLB RET
DM0:[30,10]AM.FLB;1 Form name? CHILA RET
DM0:[30,10]AM.FLB;1 Form name? RET
DM0:[30,10]AM.HLP;1 Form name? CHI001 RET
DM0:[30,10]AM.HLP;1 Form name? RET
FUT>
```

The following sections describe each section of the output file that the Form Utility creates when you use the /FD option.

3.5.3.5 The /FD Option: Form Description Header — Figure 3-4 shows an example of the form description header in a printable form description.

```
Form name:              BYE
Help form name:         BYEHLP
First line:             1
Last line:              23
Date created:           29-DEC-79
Owner ID:               0
Form length:            1478 bytes
Number of fields:       4
Impure area size:       2010 bytes
```

Figure 3-4. The /FD Option: The Form Description Header

The individual lines in the form description header provide the following information:

- Form name

  As assigned by completing the Form-Wide Attributes Questionnaire in the Form Editor.

- HELP form name

  As assigned by completing the Form-Wide Attributes Questionnaire in the Form Editor.

- First line

  As assigned by completing the Form-Wide Attributes Questionnaire in the Form Editor.

- Last line

  As assigned by completing the Form-Wide Attributes Questionnaire in the Form Editor.

- Date created

  The most recent date on which the form was processed with the Form Editor.

- Owner ID

  Reserved for future use.

- Form attributes

  If the reverse screen, current screen, and wide screen attributes have been selected in the Form-Wide Attributes Questionnaire, they are reported on this line.

- Form length

  As reported in the Form-Wide Attributes Questionnaire in the Form Editor.

- Number of fields

  The number of fields with different names. Each occurrence of a scrolled or indexed field is counted.

- Impure area size

  As reported in the Form-Wide Attributes Questionnaire in the Form Editor.

3.5.3.6 The /FD Option: The Image Map — Figure 3-5 shows an example of the 80-column image map that the Form Utility produces in a printable form description. (Although the Form Utility shows all 24 lines in the image map, the figure has been compressed for printing in this manual.)

```
            1         2         3         4         5
   12345678901234567890123456789012345678901234567890123456 78
 1:
 2:
 3:
 4:                            CUSTOMER PROFILE
 5:
 6:
 7:
 8:
 9:                 Annual Income in Thousands   $000000
10:                 Expected Purchases           $000000
11:                 Number of employees           000000
12:
13:
```

Figure 3-5. The /FD Option: The Image Map

For 80-column forms, the borders of the image map include scales that show the line and column numbers for the map. For 132-column forms, the line numbers do not appear. Except for the video attributes of the form, the image map shows the form as the operator will see it before the Form Driver or the operator enters information in any fields. Each character of the constant text appears in the correct line and column position. Each field appears in the image map with the clear character that was assigned by using the Form Editor, and each field includes any field-marker characters, such as the hyphen (-).

3.5.3.7 The /FD Option: The Video Attributes Map — Figure 3-6 shows an example of the 80-column video attributes map that the Form Utility produces in a printable form description. (Although the Form Utility shows all 24 lines in the map, the figure has been compressed for printing in this manual.)

```
            1         2         3         4         5
   1234567890123456789012345678901234567890123456789012345678
 1:
 2:
 3:                          22222222222222222222
 4:                          22222222222222222222
 5:                          22222222222222222222
 6:
 7:
 8:
 9:                00000000000000000000000000000000222222
10:                0000000000000000000            0222222
11:                00000000000000000000            222222
12:
13:
```

Key to Video Attributes

```
Code      Attributes
0         Normal
2         Reverse Video
4         Bold
6         Bold, Reverse Video
```

Figure 3-6. The /FD Option: The Video Attributes Map

For 80-column forms, the borders of the video attributes map include scales that show the line and column numbers for the map. For 132-column forms, the line numbers do not appear. Within the map, a one-digit or one-letter code for the video attributes of each character appears at the character's position. The codes that appear in each map are listed at the bottom. Table 3-2 contains a complete list of all the codes and their meanings. In each map, the codes that actually appear are described below the map under the heading "Key to Video Attributes."

Table 3-2. **The /FD Option: Video Attributes Codes and Meanings**

Code	Meaning
0	Normal
1	Underline
2	Reverse video
3	Reverse video, Underline
4	Bold
5	Bold, Underline
6	Bold, Reverse video
7	Bold, Reverse video, Underline
8	Blinking
9	Blinking, Underline
A	Blinking, Reverse video
B	Blinking, Reverse video, Underline
C	Blinking, Bold
D	Blinking, Bold, Underline
E	Blinking, Bold, Reverse video
F	Blinking, Bold, Reverse video, Underline

3.5.3.8 The /FD Option: Field Descriptions — Figure 3-7 shows an example of the field description that the Form Utility produces in the printable form description.

```
1 39      Field INV    of length 10
          Display attributes: Autotab, Vertical
          Field Type: Numeric, Scrolled, Indexed and repeated
          6 times
          Clear characters: "@'
          Help text: "Invoice number is required information'
          Picture value: "99999999.99'
```

Figure 3-7. The /FD Option: Field Descriptions

The individual lines in each field description provide the following information:

- Field name, size, and position

  The first line of the field description describes the starting position of the field in terms of the row and column numbers for the first character ("1 39" in the example above). The line also provides the field name and the length of the field as used by the application.

- Display attributes

  Any of the following field attributes, as assigned with the Form Editor:

  - Autotab
  - No Echo
  - Display Only
  - Right Justify
  - Fixed Decimal
  - Zero Fill
  - Zero Suppress
  - Uppercase
  - Must Fill
  - Response Required
  - Supervisor Only
  - Clear Char
  - Vertical (indexed)
  - Horizontal (indexed)

- Field type

  The type of characters that an operator can enter in the field, corresponding as follows with the field picture characters that the Form Editor accepts:

  - 9 Numeric type
  - A Alphabetic type
  - C Alphanumeric type
  - N Signed numeric type
  - X Any printing character

The other field type features listed in this section are:

- Indexed
- Mixed picture
- Scrolled

- Clear character

  As assigned with the Form Editor Field Attributes Questionnaire.

- Help text

  As assigned with the Form Editor Field Attributes Questionnaire.

- Picture value

  As entered in the Form Editor's FIELD mode.

3.5.3.9 The /FD Option: The Named Data Map — Figure 3-8 shows an example of the named data map that the Form Utility produces in the printable form description.

```
                Named Data Information


Name            Data

ONE             216 295   23    6 171    93 END
ONEOUT          DM2:[2,60]BYWAYS.DAT
```

Figure 3-8. The /FD Option: The Named Data Map

Chapter 4
Introduction to the FMS-11 Form Driver (FDV)

The Form Driver is a library of routines that is a subcomponent of your program. In an application that uses video images of forms on the terminal screen, using the Form Driver can reduce your programming effort by manipulating the screen, checking responses that an operator types, and displaying HELP messages and forms when the operator requests them.

This chapter discusses how the Form Driver interacts with:

- The form description, which is created with the Form Editor.
- The terminal operator, who enters information into the fields in a displayed form.

This chapter provides general programming requirements and mentions (but does not describe in detail) specific subroutine calls. Chapter 5 covers the programming requirements for each of the high-level languages and MACRO-11 in detail. Chapter 6 contains an alphabetical list of the calls and gives a full description of each one. Chapter 7 describes programming techniques for typical Form Driver applications.

4.1 Form Driver Interaction with the Form Description

This section provides a general description of how the Form Driver uses forms to display information for the operator, how it guides the operator through a form, and how it collects the responses the operator types. Throughout the section, and in many of the other descriptions in this and later chapters, the term "form" refers to the image the operator sees and to the computerized form description the Form Driver manipulates internally.

4.1.1 Media-Resident and Memory-Resident Forms

Your program can use form descriptions in two ways:

- As media-resident forms, by reading them directly from a form library file stored on a mass storage volume, such as a disk.
- As memory-resident forms, for which the form descriptions are included with the program itself as a part of the task-building procedure.

Both ways use form descriptions that have been created with the Form Editor and processed with the Form Utility. For example, after using the Form Editor to create a form description, you must use the Form Utility to store the description in a form library file or to produce the object module that memory-resident usage requires. Chapter 2 describes how to use the Form Editor, and Chapter 3 describes how to use the Form Utility. Chapter 5 describes the task-building procedure for each language.

For each call to display a form, the Form Driver first checks the set of memory-resident forms. When memory-resident and media-resident form descriptions have the same form name, the Form Driver uses only the memory-resident version.

4.1.2 Defining Forms and Fields by Name

To read the form from its form library file or find its memory-resident description, the Form Driver needs only the name assigned to that form by the Form Editor. To find a field within a form, the Form Driver requires only the name assigned to that field, regardless of where the field is located within that form. As long as changes to form and field characteristics have no effect on the logic of your programs, these characteristics can be changed without program modification.

4.1.3 Displaying the Form

A typical procedure for displaying a form at the beginning of an FMS application follows. (The calls are described in full in Chapter 6, and the MACRO-11 procedures are explained in Chapter 5.)

1. If your program uses media-resident forms:
   - Identify the I/O channel the Form Driver is to use for reading form descriptions from the form library file. With high-level languages, use the FLCHAN call. With MACRO-11, complete the Required Argument List.
   - Open the form library file using the FLOPEN call.
2. For all applications, identify an internal storage area, called the impure area, that the Form Driver is to use for field values and other form requirements. With high-level languages, use the FINIT call. With MACRO-11, complete the Required Argument List.

3. Display a form. Use the FCLRSH or FSHOW call.

   The Form Driver provides two calls for displaying a form: FCLRSH and FSHOW. The FCLRSH call clears the entire terminal screen before displaying a form. The FSHOW call clears only the screen lines that are required by the form you want to display. If you use short forms, you can use the FSHOW call to create a screen display for the operator that is composed of more than one form or part of a form. In this case, only one form would normally be active for the operator, but you could also use special techniques like the ones described in Chapter 7 to keep more than one form active at the same time.

4.1.4 The HELP Function

Whenever your program issues a call for an operator response, the Form Driver can display two levels of help if the operator requests it: help for the field in which the cursor is located and help for the entire form. When the operator uses the Form Driver's HELP function once, the Form Driver displays the help text that was typed in the Field Attribute Questionnaire. When the operator uses the HELP function again, the Form Driver displays the HELP form that was specified in the Form-Wide Attributes Questionnaire.

The operator can erase any HELP form and have the Form Driver restore the original form at any time. The position of the cursor in the original form and all field values will be unchanged.

For each form in your application, both the help text for fields and the HELP forms have to be specified when the form is created or changed with the Form Editor. For applications that use media-resident forms, the HELP forms must be stored in the same form library file as the forms with which they are associated.

4.1.5 Internal Storage of Field Values - The Impure Area

When a form is displayed, its form description is stored internally by the Form Driver in a special area called the impure area. Internal buffers are set up for the fields in the display and for the other characteristics of the form, such as named data labels and values. When you issue calls to get or display values, identify fields, and complete other processes, the Form Driver uses and updates the impure area, it displays information for the operator, and it provides values to your program.

Both the Form Editor and the Form Utility report the size of the impure area needed for each specific form for a MACRO-11 application. For high-level language programs, the impure area requires an additional 64 bytes. Although you can vary the size of the impure area to match the exact needs of each form, for practical purposes you need only define one impure area large enough to accommodate the largest form your application uses. With high-level languages, use the FINIT call to define the impure area. With MACRO-11, complete the Required Argument List.

The impure area is used only by the Form Driver. Your program cannot use it directly. You can, however, use the FRETN and FRETAL calls to determine any field value stored in the impure area. The FRETN call returns the value for a specified field, and the FRETAL call returns a concatenated string of values for all fields.

4.1.6 Guiding the Operator Responses

In guiding the operator from field to field, the Form Driver moves as follows: first, from left to right within a screen line, then line by line from the top of the form to the bottom. On the other hand, the field attributes, such as the Vertical Indexed attribute and your call order, usually define a unique sequence for each form. In fact, your program is entirely in control of the order in which the operator works with the fields.

For example, you can control the order completely by using only the FGET call to get the value of a specified field. By repeating the call and specifying different fields, you require the operator to complete the fields in the order you specify.

You can also allow the operator partial control by using the FGETAF call. This call allows the operator to choose any field in the form. The operator can respond in only one field, but this can be any of the non-display-only fields in the form. Since this call also identifies the name of the completed field, your program can then direct the operator to any other field.

The call for all field values, the FGETAL call, gives the operator complete control over the order in which the fields are completed. The Form Driver returns the field values to your program and updates the impure area only when the operator signals the entire form is complete.

4.1.7 The Order in Which the Form Driver Concatenates Fields

Two of the calls for operator responses get more than one field value. The FGETAL call gets a concatenated string of all field values for the form. The FINLN call gets a concatenated string of the field values in one line of a scrolled area. Regardless of the order in which the operator has entered and corrected the field values, the Form Driver concatenates them according to the following conventions:

1. Except for fields that have the Vertical Indexed attribute, field values are concatenated from left to right within each line and then line-by-line from the top of the form to the bottom. This convention includes fields with the Horizontal Indexed attribute and fields in scrolled areas.
2. Fields with the Vertical Indexed attribute are concatenated line-by-line (in index order).
3. Within the concatenated string, the length of each field value is the full length of the field. Each value shorter than the field is padded out to the field length with the fill character assigned to the field. For a Right Justified field, the fill characters precede the value; for a Left Justified field, they follow the value.

   Two calls display more than one field value. The FPUTAL call displays all field values in the form. The FOUTLN call displays the field values for one line of a scrolled area. For these calls, you must create a concatenated string of the values, including fill characters, where they are needed for padding. The values must be in the same order the Form Driver would use in processing the call to get all field values.

4.1.8 Text, Field-Marker Characters, and Video Attributes

After displaying a form, the Form Driver normally uses only the information that relates to the fields, such as a field picture, the fill and clear characters, the default value, and the line of HELP information. Unless the operator uses the CTRL/W function to have the Form Driver redisplay the entire form, the Form Driver makes no further use of information not related to the fields, such as the text in the form, the field-marker characters, and the video attributes of the characters displayed.

In particular, the field values the Form Driver returns do not contain any of the field-marker characters the operator sees, such as the hyphen (-), decimal point (.), slash (/), and minus sign (–). Also, the field values your program passes to the Form Driver to display must not include field-marker characters.

4.1.9 Processing Fields

This section describes how the Form Driver processes fields in terms of the field attributes.

4.1.9.1 The Field Pictures — The Form Driver uses field pictures only when the operator is typing field values. The values your program passes to the Form Driver for display are not validated against the field pictures.

When the operator is responding, a field picture is used to:

- Validate that each character satisfies the requirements of the picture character at the corresponding position. For example, in a field with the mixed picture 999AAA, the Form Driver accepts only digits in the first three positions and only letters in the last three positions.
- Limit the operator's use of the INSERT and OVERSTRIKE modes of entering field values. For example, the operator cannot change the combination of modes used for a fixed-decimal field or use the INSERT mode when he or she is completing a field that has a mixed picture. (Section 4.2.3 describes the INSERT and OVERSTRIKE modes in detail.)

Section 2.6.7 describes the field picture characters and the valid operator responses in detail.

4.1.9.2 The Right Justified and Left Justified Field Attributes — The Form Driver uses the Right Justified and Left Justified attributes to:

- Determine the position of the cursor when it is first displayed in a field. (Section 4.2.2.2. describes this position, called the initial position of the cursor, in detail.)
- Align the field value both on the screen and in the impure area when the value is shorter than its field. For example, the value in a Right Justified field always ends at the last character position in the field.
- Determine when the operator has filled the field if the field has the Autotab attribute. (Section 4.1.8.5 describes in detail the effect of the Autotab attribute.)
- Set the default mode of entering values in the field. For example, the INSERT mode is the default for a Right Justified field. (Section 4.2.3 describes in detail the INSERT and OVERSTRIKE modes.)

4.1.9.3 The Clear Character and Zero Fill Attributes — The Clear Character and Zero Fill attributes affect how field values are padded on the screen and in the impure area. The clear character is displayed and the fill character is used as padding in the impure area. When a field has no value, it is displayed with only the assigned clear character; it is stored in the impure area with only the assigned fill character.

If a field has the Zero Fill attribute, the clear character must be zero (0), and, if necessary, the field value is padded with zeroes in the impure area. The Form Editor does not allow other combinations. If a field does not have the Zero Fill attribute, the clear character can be any printing character, and, if necessary, the field value is padded with spaces in the impure area.

4.1.9.4 The Default Value — When the Form Driver displays a form, it displays the default field values and stores them as the current field values in the impure area. However, neither the Form Editor nor the Form Driver validates the default value in any way. For example, the Form Editor does allow you to assign the numeric default value 13467 for a field with the picture AAAAA, and the Form Driver does display the value in such a case, even though the Form Driver does not allow the operator to enter the value. Therefore, when you develop your application, you must make sure the default value is correct for the field.

4.1.9.5 The Autotab Attribute — When the operator types the character that fills a field with the Autotab attribute, the Form Driver terminates the field as if the operator had pressed the TAB key. (Section 4.2.4 describes the use of the TAB key and the other field terminators in detail.)

With respect to the Autotab attribute, the Form Driver determines that a field has been filled as follows:

- The Must Fill attribute, if assigned to the field, must be satisfied. (Section 4.1.8.6 describes the Must Fill attribute in detail.)
- For a Left Justified field, the operator must have typed a character in the rightmost character position.
- For a Right Justified field, the leftmost character position must contain a character other than the Fill Character.

4.1.9.6 The Response Required and Must Fill Attributes — The Form Driver uses the Response Required and Must Fill attributes to validate the completeness of an operator's response in a field.

In a field with the Response Required attribute, the operator must type at least one character other than the assigned fill character.

In a field with the Must Fill attribute, the operator can type nothing or fill the field completely. The Form Driver will not accept a field value shorter than its field or a value that contains a fill character.

The Form Driver validates the Response Required and Must Fill attributes at different times depending on the call your program issues for an operator response. For the call to get all field values from the operator, the Form Driver validates these attributes for each field:

- When the operator terminates input in the field with the TAB key.
- When the operator signals completion of the form by pressing the ENTER or RETURN key.

For the other calls, the Form Driver validates these attributes when the operator terminates the field. (Section 4.2.4 describes in detail how to use the ENTER and RETURN keys to terminate a field and how to use the other field terminators.)

4.1.9.7 The Fixed Decimal Attribute — The Form Driver uses the Fixed Decimal attribute to:

- Determine the position of the cursor when it is first displayed in a field. This position is called the initial cursor position. (The initial cursor position is described in detail in Section 4.2.2.2.)
- Align the parts of the field value to the left and right of the decimal point. For example, the Form Driver displays the part to the left of the decimal point as a Right Justified field and the part to the right of the decimal point as a Left Justified field.
- Determine the fill and clear characters for the left and right parts of the field. For example, the Form Driver always displays the decimal part of the field value as a Zero Fill and Left Justified field, regardless of whether the Zero Fill or Clear Character attribute is assigned. The Form Driver applies the assigned Zero Fill or Clear Character attribute to the part of the value to the left of the decimal point only.

Section 4.2.5 describes the special characteristics of fixed-decimal fields with respect to typing and editing the field values.

4.1.9.8 The Horizontal and Vertical Indexed Attributes — The Form Driver uses the Horizontal and Vertical Indexed attributes to:

- Define the indexes for the individual fields that make up the indexed field.
- Move the cursor to the proper individual field when your program issues a call for the value of a specific indexed field.

  For example, when your program issues the FGETAL call to get all field values, the TAB function moves the cursor through an indexed field as shown in Figure 4-1. (Section 4.2.4 describes the TAB function and the other field terminator functions in detail.)
- Determine the order in which the field values are concatenated for the call to get all field values. Figure 4-1 illustrates the concatenated order.

4.1.9.9 The Display Only Attribute — The Form Driver uses the Display Only attribute to allow your program to display variable field values without allowing the operator to type or change the field values. The Form Driver does not allow the operator to position the cursor in a display-only field. When the operator uses the TAB function or other functions to move the cursor from field to field, the cursor jumps past the display-only fields as if they were part of the form's background text. (Section 4.2.4 describes the TAB function and the other field terminator functions in detail.)

4.1.9.10 The Echo Off Attribute — The Form Driver uses the Echo Off attribute to prohibit field values from being displayed in fields. When the operator responds in an Echo Off field or when your program issues a call to display a field value in an Echo Off field, the Form Driver returns the field value to your program and stores it in the impure area but does not display the field value.

4.1.9.11 The Supervisor Only Attribute — When your program uses the FSPON call to turn on the supervisor-only mode, the Form Driver prevents the operator from typing or changing values for fields that have the Supervisor Only attribute. In effect, after the program issues the FSPON call the Form Driver treats all fields with the Supervisor Only attribute as display-only fields.

This treatment, which remains in effect until the program issues the FSPOFF call, applies to all forms displayed. When the program issues the FSPOFF call, the Form Driver ignores the Supervisor Only attribute until the program issues the FSPON call again.

4.1.9.12 The Scrolling Attributes — Although the Form Editor and Form Driver allow a form length of 23 lines or less, both FMS components allow the definition of multiline sections within a form for displaying data tables of as many lines as are needed. Each of these sections is called a scrolled area because it can be used to treat a long data table like a scroll, winding it up or down to expose in the scrolled area the specific lines you want the operator to see or complete. In effect, a scrolled area is like a separate window within a form and the terminal screen. In the window, you can show any part of a longer file.

Each scrolled area must be at least two lines long. Within one form you can define as many separate scrolled areas as will fit within 23 lines. Each line can have as many separate fields as will fit on one screen line, but for each scrolled area, all lines must be identical with respect to the number of fields, their sizes and attributes, and all other details.

Because the Form Driver can store field values only for the fields on the terminal screen, your program must maintain all scrolled area field values not displayed, that is, all of the values "above" and "below" each scrolled area. When your program scrolls the lines of a scrolled area up or down, the program must collect the line of values, if any, scrolled into the area.

Chapter 7, Form Driver Programming Techniques and Examples, includes detailed explanations of scrolled area usage and summaries of typical programming methods applicable to scrolled areas.

4.2 Form Driver Interaction with the Terminal Operator

While working with an FMS application, the terminal operator might feel he or she is always in control of the form displayed on the screen. In fact, the operator has no control until your program permits it by issuing one of the following Form Driver calls for an operator response:

- FGET, to get the value of a specified field.
- FGETAF, to get the value of the field the operator chooses.
- FGETAL, to get a concatenated string of all field values for the current form.
- FINLN, to get a concatenated string of all the field values for a line of a scrolled area.

Each of these four calls puts the operator in control until the requirements of the call have been satisfied. For example, after your program issues the FGET call for the value of a specific field, the operator can type and correct the response for as long as he or she wishes. The operator can also request help by using the HELP function. When the operator terminates the field with a field terminator function such as the TAB function, the Form Driver returns control to the program. Then, until the the program issues another call for an operator response, the operator has nothing to do.

This section introduces the three general kinds of operator activity:

- Correcting errors and requesting HELP.
- Editing fields.
- Terminating and choosing fields.

4.2.1 Signaling and Recovering from Errors

The Form Driver responds to typographical errors and invalid use of editing and field termination functions as follows:

- For all errors, the Form Driver rings the terminal bell and ignores the invalid character or function.
- For some errors, the Form Driver also displays a one-line explanation on the bottom screen line. For example, when an operator tries to enter a letter in a field that has been designed to accept only numbers, the Form Driver rings the bell and displays the following message:

  ```
  NUMERIC REQUIRED
  ```

Appendix E lists and explains each of the messages that can appear in these cases.

The Form Driver also provides a special operating mode, called the debug mode, which produces an extensive set of error messages useful to you while you are developing and refining your FMS application programs. If an error occurs while you are using the Form Driver with the debug mode feature, the Form Driver stops your program, rings the terminal bell, displays the debug mode message on the bottom screen line, and then waits for you to press the ENTER or RETURN key before resuming your program. (Section 5.1 describes the debug mode in detail.)

4.2.1.1 The HELP Key and Help Messages — The HELP function can display two levels of information.

When the operator presses the HELP key for the first time, the Form Driver determines whether a help message exists for the current field. If such a one-line help message exists, the Form Driver displays it in the last line of the screen. The cursor remains in place within the field.

If the operator does not find the one-line help message sufficiently helpful, he or she may press the HELP key a second time. The Form Driver then determines whether a help form exists for the current form.

If a help form exists, the Form Driver displays the help form while saving the context of the current form. Each help form can have yet another help form associated with it. Until the last of the chain of help forms is displayed, the HELP function causes the next form in the chain to appear.

To return to the form he or she was originally working on, the operator presses the ENTER key. In response, the Form Driver restores the form and cursor as they were before the HELP key was pressed.

If no one-line HELP message exists for a particular field, the Form Driver displays the HELP form directly. When no HELP is available, the Form Driver displays a message to that effect on the last line of the screen. When, in the course of continuing work on the form, the operator types a field terminator, the Form Driver clears the last line.

4.2.1.2 Messages Controlled by the Program — There are two cases in which the Form Driver cannot distinguish between valid and invalid operator responses.

1. Although the Form Driver accepts only the operator responses that meet the requirements of the field picture that was assigned with the Form Editor, the Form Driver cannot detect a field value that is invalid in your application.
2. When an operator uses certain functions to terminate work within a field, the Form Driver waits for your program to respond rather than processing the terminator automatically.

For both of these cases, you can design the program to detect errors and other conditions and display messages to the operator. Chapter 5 describes the typical processes and techniques in detail.

4.2.1.3 Repainting the Screen: The CTRL/W Function — VT200 Keys: Hold down the CTRL key and at the same time press the W key on the keyboard.

The CTRL/W function repaints the current form and its current field values on the screen. The function is useful for ensuring that the screen is displaying the field values stored in the impure area and the background text for the form.

If part or all of an earlier form was left on the screen when the current form was displayed, the CTRL/W function erases the earlier form completely and repaints only the current form. (The FSHOW call can be used to leave an earlier form on the screen when another form is displayed.)

The CTRL/W function is always valid.

4.2.2 Field Editing Principles and Functions

Table 4-1 summarizes the field editing functions the Form Driver provides and lists the keys that control the functions. These functions are executed entirely by the Form Driver. You can implement added functions within your program by switching the LK201 keyboard of the VT200 to the alternate keypad mode and using the numeric keypad keys to control the functions you design. Keys followed by an asterisk (*) denote changes from the VT100. Appropriate VT100 keys and functions follow this table. Keys without an asterisk (*) are the same for the LK201 and the VT100 keyboards.

Unless otherwise indicated, VT200 keys referred to in Table 4-1 are identical to the corresponding VT100 keys.

Table 4-1. Field Editing Keys, Functions, and Usage for the Form Driver

Key	Function	Usage
LEFTARROW (←)	Cursor Left	Moves the cursor to the preceding position within the field.
RIGHTARROW (→)	Cursor Right	Moves the cursor to the next position within the field.
DELETE	Erase Character	In the INSERT mode, erases the character to the left of the cursor and closes the space.
		In the OVERSTRIKE mode, moves the cursor to the preceding character position within the field but erases it only when the character is the last one in a Left Justified field.
F12*	Backspace	Moves the cursor to the initial position of the previous non-display-only field.
F13*	Erase Field	Erases the entire field.
PF1	INSERT/ OVERSTRIKE	Switches from the INSERT mode to the OVERSTRIKE mode, or vice versa.
HELP*	HELP	First, displays the HELP text for the cursor's field, and then displays successive HELP forms for the current form.
CTRL/W	Repaint Screen	Repaints the screen with the current form, field values, and cursor location.
Most keyboard keys	Insertion	The keys for the printing characters on the keyboard insert their characters. In the normal (numeric) keypad mode, the numeric and punctuation keys on the keypad also insert their characters.
The following functions are for VT100 keyboards only:		
LINEFEED	Erase Field	Erases the entire field.
BACKSPACE	Backspace	Moves the cursor to the initial position of the previous non-display-only field.
PF2	HELP	First, displays the HELP text for the cursor's field, and then displays successive HELP forms for the current form.
The following control characters are for any keyboard:		
CTRL/H	Backspace	
CTRL/I	Tab	
CTRL/J	Linefeed	

4.2.2.1 Terminators and Alternate Keypad Mode — The terminal can be set to an alternate keypad mode or a normal (numeric) keypad mode, as described in the *Terminal Programmer's Reference*. The Form Driver does not change the keypad mode at any time, and, regardless of the terminal's keypad mode, the editing and terminator functions are always the same. If your program requires either of the keypad modes, you must set the mode from within the program. Section 5.2 provides further information about setting the alternate keypad mode so the numeric keys on the keypad can be used as special field terminators.

4.2.2.2 The Cursor's Initial Position in a Field — The location of the cursor when it is first displayed in a field is called the initial cursor position. The initial position depends on whether the field has the Right Justified, Left Justified, or Fixed Decimal attribute.

For Right Justified fields, the initial position is just to the right of the last character position in the field. This position is called the hanging cursor position, because the cursor hangs off the end of the field.

For Left Justified fields, the initial position is the leftmost character position in the field.

For fixed-decimal fields, the initial position is the decimal point. Section 4.2.5 describes the special characteristics of fixed-decimal fields with respect to typing and editing field values.

4.2.2.3 Inserting a Field Value: The Default Function —

VT200 Keys: The Form Driver accepts the standard letters, numbers, and special characters on the keyboard that meet the requirements of the field.

For the keyboard keys, insertion of values in fields is the default function. For the numeric and punctuation keys on the keypad, insertion is also the default when the keypad is in the normal (numeric) mode. In both cases, the operator types values as if he or she were using a typewriter.

Insertion is invalid only when it does not meet the field's requirements. For example, letters are invalid where numbers are required, and for a field that does not have the Autotab attribute, all characters are invalid when the field is full.

4.2.2.4 Erasing a Character: The DELETE Function —

VT200 Key: The DELETE key on the keyboard.

The DELETE function normally erases the character to the left of the cursor. The function has different effects, however, in the INSERT and OVERSTRIKE modes. The modes are explained in detail in Section 4.2.3.

In the INSERT mode, the Form Driver erases the character to the left of the cursor and closes up the space. In a Left Justified field, the value remains left-justified; in a Right Justified field, the value remains right-justified.

In the OVERSTRIKE mode, the DELETE function always moves the cursor one character to the left. However, to prevent an operator from accidentally introducing errors in a field with a mixed picture, the function does not erase a character in OVERSTRIKE mode except for the rightmost character in a Left Justified field.

The DELETE function is invalid when the cursor is on the leftmost character in the field.

4.2.2.5 Erasing a Field: The LINEFEED Function —

VT100 Key: The LINEFEED key on the keyboard.
VT200 Key: The F13 key on the LK201 keyboard.

Regardless of the cursor's position in a field, the LINEFEED function erases all characters (except field-marker characters) in the field. The Form Driver then displays the assigned clear character for the field and in the impure area fills the field with the assigned fill character. When the function is complete, the cursor is located at the initial position for the field (the leftmost character for a Left Justified field and to the right of the rightmost character for a Right Justified field).

The LINEFEED function is always valid input in a field.

4.2.2.6 Moving the Cursor to the Right: The RIGHTARROW Function —

VT200 Key: The RIGHTARROW key (→) on the keyboard.

The RIGHTARROW function normally moves the cursor one character to the right within a field. However, the cursor always skips the field-marker characters, such as the hyphen (-) and slash (/) in a field.

The RIGHTARROW function is invalid when the cursor is to the right of the rightmost character in a field.

4.2.2.7 Moving the Cursor to the Left: The LEFTARROW Function —

VT200 Key: The LEFTARROW key (←) on the keyboard.

The LEFTARROW function normally moves the cursor one character to the left within a field. However, the cursor always skips the field-marker characters in a field.

The LEFTARROW function is invalid when the cursor is on the leftmost character of a field.

4.2.3 Switching the Insertion Modes: the INSERT/OVERSTRIKE Function

VT200 Key: The PF1 key on the keypad.

While the operator is typing a field value, the INSERT and OVERSTRIKE insertion modes control how the Form Driver displays the characters. For most of the different types of fields that can be designed, the operator can control the insertion mode by using the INSERT/OVERSTRIKE function.

When either the operator or your program first moves the cursor to a field, the Form Driver sets the insertion mode according to the attributes of the field. The INSERT mode is the default for Right Justified fields, and the OVERSTRIKE mode is the default for Left Justified fields.

While the operator types in INSERT mode in a Left Justified field, the Form Driver inserts each character at the position of the cursor. The cursor, the character at the cursor, and all characters within the field to the right of the cursor are shifted to the right. In a Right Justified field, all characters to the left of the cursor are shifted to the left and the new character is inserted directly to the left of the cursor.

In the OVERSTRIKE mode, the Form Driver replaces the character at the cursor with the character typed and moves the cursor one character to the right.

The INSERT/OVERSTRIKE function switches the insertion mode from one to the other. For example, when the Form Driver is initially in the INSERT mode, by pressing the PF1 key once, the operator switches the Form Driver to the OVERSTRIKE mode; by pressing PF1 again, the operator switches the Form Driver back to the INSERT mode.

In fields with mixed pictures, the INSERT mode is invalid. In fixed decimal fields, the INSERT/OVERSTRIKE function is ignored entirely because of the special data entry conventions fixed-decimal fields require. (Section 4.2.5 describes how the Form Driver handles operator responses in fixed-decimal fields.) In all other cases, the INSERT/OVERSTRIKE function is valid.

4.2.4 Field Terminating Functions

The operator uses the field terminating functions for the Form Driver to signal that he or she wishes to work with a different field or a different form. The Form Driver processes these functions differently depending on what Form Driver call is currently being executed. In many cases, the Form Driver gives your program an opportunity to intercept and change the terminator function the operator has used. The Form Driver identifies each terminator function with a unique terminator code.

This section describes the normal effects of the terminator functions and lists the terminator codes for both high-level languages and MACRO-11. (Section 5.2 describes the field terminator processing in detail, including use of the terminator codes.)

4.2.4.1 Signaling the Form Is Complete: The ENTER and RETURN Functions —

VT200 Keys: The ENTER key on the keypad and the RETURN key on the keyboard both control this terminator.

Terminator Code and Value:

MACRO-11 Code:	FT$NTR
High-Level Language Value:	0

The ENTER and RETURN functions signal that the operator has completed the current form. The operator uses either function when he or she does not want to enter or change any field values.

When an FGETAL call is issued, the Form Driver does not accept either the ENTER or the RETURN function until all field values satisfy their field requirements. For example, a Response Required field must have a response and a Must Fill field must be filled. However, the Form Driver cannot determine if the field values are valid. Your program must do this.

For any other Form Driver call, control is returned to the program if the requirements for the current field value are satisfied.

4.2.4.2 Moving the Cursor to the Next Field: The TAB Function —

VT200 Key: The TAB key on the keyboard.

Terminator Code and Value:

MACRO-11 Code:	FT$NXT (when terminating a field outside a scrolled area)
	FT$SNX (when terminating the last field in a line within a scrolled area)
High-Level Language Value:	1 (when terminating a field outside a scrolled area)
	6 (when terminating a field within a scrolled area)

The TAB function is valid only when the requirements for the current field value (Response Required and/or Must Fill) are satisfied.

The effects of the TAB function depend on what Form Driver call is being executed.

For the FGETAL and FINLN calls and for the FGETAF call before the operator enters or changes a field value, the Form Driver processes the function directly and moves the cursor to the initial position of the next modifiable field. (Section 4.1.6 describes the order in which the Form Driver moves from field to field.)

For the FGET call and for the FGETAF call after the operator enters or changes a field value, the Form Driver transfers control to the program. The next call in your program determines what the operator sees. For example, after the operator terminates a field with the TAB function, your program can display a new form, calculate and display a value in a display-only field, or issue another call for another operator response in a specific field.

The function is invalid when the cursor is in the last non-display-only field of the form.

4.2.4.3 Moving the Cursor to the Previous Field: The BACKSPACE Function —

VT100 Key: The BACKSPACE key on the keyboard.
VT200 Key: The F12 key on the LK201 keyboard.

Terminator Code and Value:

MACRO-11 Code:	FT$PRV (when terminating a field outside a scrolled area)
	FT$SPR (when terminating the first field in a line within a scrolled area)
High-Level Language Value:	2 (when terminating a field outside a scrolled area)
	7 (when terminating a field within a scrolled area)

The effects of the BACKSPACE function depend on what Form Driver call is being executed.

For the FGETAL and FINLN calls, and for the FGETAF call before the operator enters or changes a field value, the Form Driver processes the function directly and moves the cursor to the initial position of the previous non-display-only field. (Section 4.1.6 describes the order in which the Form Driver moves from field to field.)

For the FGET call, and for the FGETAF call after the operator enters or changes a field value, the Form Driver transfers control to the program. The next call in your program determines what the operator sees.

The function is invalid when the cursor is in the first non-display-only field of the form.

4.2.4.4 Scrolled Area Moves: The UPARROW Function —

VT200 Key: The UPARROW key (↑) on the keyboard.

Terminator Code and Value:

MACRO-11 Code:	FT$SBK
High-Level Language Value:	9

The UPARROW function is valid only when the cursor is in a field within a scrolled area. This function always transfers control to your program. Therefore, you can use the function any way you wish, and the effects the operator sees depend on the next calls your program issues.

The Form Driver processes the UPARROW function only when you specify its code in the FPFT call. The Form Driver either moves the cursor to the preceding data line within the scrolled area and places the cursor at the initial position of the first non-display-only field in that data line, or scrolls the area backward and places the cursor at the initial position of the first non-display-only field in the current line.

When the cursor is in the top screen line of the scrolled area, or if the program specifies data to update the top line, the UPARROW function scrolls the bottom line of information off the screen, scrolls a new line of information into the top scrolled line, and moves the intermediate scrolled lines downward. If the cursor is in the top line and your program specifies values for the new line of information, these values are displayed; otherwise, the default field values are displayed.

The function is valid only when the cursor is in a field within a scrolled area.

4.2.4.5 Scrolled Area Moves: The DOWNARROW Function —

VT200 Key: The DOWNARROW key (↓) on the keyboard.

Terminator Code and Value:

MACRO-11 Code:	FT$SFW
High-Level Language Value:	8

The DOWNARROW function is valid only when the cursor is in a field within a scrolled area. The function always transfers control to your program. Therefore, you can use the function any way you wish, and the effects the operator sees depend on the next calls your program issues.

The Form Driver processes the DOWNARROW function only when you specify its code in the FPFT call. The Form Driver either moves the cursor to the next data line within the scrolled area and places the cursor at the initial position of the first non-display-only field in that data line, or scrolls the area forward and places the cursor at the initial position of the first non-display-only field in the current line.

When the cursor is in the bottom screen line of the scrolled area, or if the program specifies data to update the bottom line, the DOWNARROW function scrolls the top scrolled line of information off the screen, scrolls a new line of information into the bottom scrolled line, and moves the intermediate scrolled lines upward. If the cursor is in the bottom line and your program specifies values for the new line of information, these values are displayed; otherwise, the default field values are displayed.

The function is valid only when the cursor is in a field within a scrolled area.

4.2.4.6 Scrolled Area Moves: The EXIT SCROLLED AREA BACKWARD Function —

VT200 Key: The PF3 key on the keypad.

Terminator Code and Value:

MACRO-11 Code:	FT$XBK
High-Level Language Value:	4

The EXIT SCROLLED AREA BACKWARD function is valid only when the cursor is in a field within a scrolled area. The function always transfers control to your program. Therefore, you can use the function any way you wish, and the effects the operator sees depend on the next calls your program issues.

The Form Driver processes the EXIT SCROLLED AREA BACKWARD function only when you specify its code in the FPFT call. The Form Driver moves the cursor to the initial position of the first non-display-only field above the scrolled area.

The function is invalid when:

1. The cursor is in a field not within a scrolled area.
2. There is no non-display-only field above the scrolled area.

4.2.4.7 Scrolled Area Moves: The EXIT SCROLLED AREA FORWARD Function —

VT200 Key: The PF4 key on the keypad.

Terminator Code and Value:

MACRO-11 Code:	FT$XFW
High-Level Language Value:	5

The EXIT SCROLLED AREA FORWARD function is valid only when the cursor is in a field within a scrolled area. The function always transfers control to your program. Therefore, you can use the function any way you wish, and the effects the operator sees depend on the next calls your program issues.

The Form Driver processes the EXIT SCROLLED AREA FORWARD function only when you specify its code in the FPFT call. The Form Driver moves the cursor to the initial position of the first non-display-only field below the scrolled area.

The function is invalid when:

1. The cursor is in a field not within a scrolled area.
2. There is no modifiable field below the scrolled area.

4.2.5 Typing and Editing Fixed-Decimal Values

The initial position of the cursor in a fixed-decimal field is the decimal point the Form Driver displays. The decimal point is a field-marker character. It is not stored in the impure area or returned to your program as part of the field value.

While typing a fixed-decimal value, the operator will observe that the Form Driver treats the area to the left of the decimal point as if it were a Right Justified field and the area to the right of the decimal point as if it were a Left Justified field. When the operator types digits into a field with the cursor at the initial position, the Form Driver displays all these digits to the left of the decimal point until the operator actually types a decimal point. Thereafter, when the operator types additional digits into the field, the Form Driver displays these digits to the right of the decimal point.

As the operator edits a fixed-decimal value, the LINEFEED function erases the entire value and leaves the cursor at the initial position. Normally, the DELETE function also erases the digits in the field value. However, with the cursor just to the right of the decimal point, the DELETE function moves the cursor back to the decimal point but does not erase it.

Chapter 5
Form Driver Programming Requirements and Concepts

This chapter provides a technical overview of the Form Driver for the high-level language and MACRO-11 programmer. (Although several sections of this chapter discuss the Form Driver calls, the principal description of each call appears in Chapter 6.)

The topics in this chapter are arranged in three general groups:

1. Information important to programmers who design or write Form Driver applications (e.g., checking the status of Form Driver calls or using field terminator features).
2. Information that applies only to the high-level languages. The sections in this group cover BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV, and FORTRAN-77. The section for each language describes the data types, syntax requirements, arguments used, and typical procedures for task-building an FMS application program.
3. Information that applies only to MACRO-11.

5.1 Features for Checking Call Status

To improve the effectiveness of FMS applications and reduce the time required for production of fully debugged applications, the Form Driver maintains the completion status of each call and provides four general ways to obtain the status:

- For high-level language applications, the FSTAT call returns the Form Driver status code for the last call processed. The FSTAT call also returns the FCS or RMS system error code when a call fails because of an error in opening or reading a form library file.
- For MACRO-11 applications, a two-word Status Block holds the Form Driver and system status codes for the last call processed.

- For added support while an FMS application is being developed, a special debug mode is available for displaying explicit messages about the status of calls.
- For customized support of FMS applications in the field, the FPUTL call can be used to signal the application operator about program conditions using any message you think appropriate.

5.1.1 Form Driver and System Status Codes

Table 5-1 lists and describes the returned status values and codes. For FMS applications in high-level languages, the FSTAT call returns one of the listed numeric codes in the first of its two status arguments. For applications in MACRO-11, DIGITAL recommends you use the listed global symbols instead of the numeric codes to ensure greater application compatibility with later versions of FMS software.

Two of the status conditions listed in Table 5-1 indicate an error in trying to open or read a form library file (code values –4 and –18). In these two cases, the FSTAT call also returns (in the second status argument) FCS or RMS system error codes that help to define the exact cause of the problem. For the full list of FCS system errors, refer to the *RSX-11M/M-PLUS I/O Operations Reference*. For RMS errors, refer to the *RMS-11 User's Guide*.

Note the status code FE$DLN, value –16., (data specified too long for output), is returned to the program only in debug mode. Regardless of whether the Form Driver provides support for debug mode, the specified data is truncated when displayed and the Form Driver completes the call in the normal way.

Table 5-1. Summary of Returned Status Values and Codes

Status Value in High-Level Languages (Decimal)	Status Code (MACRO-11)	Meaning
>0	>0	Successful completion of call.
1.	FS$SUC	Successful completion.
2.	FS$INC	Current form incomplete.
– 1.	FE$FCD	Specified function code undefined.
– 2.	FE$IMP	Impure area too small.
– 3.	FE$FSP	Invalid file specification.
– 4.	FE$IOL	Error encountered opening form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the Status Block).
– 5.	FE$FLB	Specified file not form library.
– 6.	FE$ICH	Invalid channel number specified.
– 7.	FE$FCH	Form library not open on specified channel.

Table 5-1. Summary of Returned Status Values and Codes (Cont.)

Status Value	Status Code	Meaning
– 8.	FE$FRM	Invalid form definition.
– 9.	FE$FNM	Specified form does not exist.
–10.	FE$LIN	Invalid first line number to display form.
–11.	FE$FLD	Specified field does not exist (invalid field name or index).
–12.	FE$NOF	No fields defined for current form.
–13.	FE$DSP	Get call illegal for display-only field(s).
–14.	FE$NSC	Specified field not in scrolled area.
–15.	FE$DNM	Named data specified does not exist.
–16.	FE$DLN	Data specified for output too long (truncated by Form Driver). This error is returned by the Form Driver only when debug mode support is included.
–17.	FE$UTR	Undefined field terminator.
–18.	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
–19.	FE$IFN	Specified call invalid in current context of form.
–20.	None	Wrong number of arguments in call. (For high-level language programs only.)
–21.	None	Impure area not yet initialized. (For high-level language programs only.)
–22.	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

5.1.2 Debug Mode Support for Application Development

The Form Driver debug mode is available to both high-level language and MACRO-11 programmers. The debug mode is a Form Driver configuration option that adds explicit messages for the status conditions to the Form Driver. Chapter 8 describes the configuration dialogue and procedure in detail. To obtain debug mode support, programmers generally configure two versions of the Form Driver: one version that includes the support and another version that does not include it. The version with the debug mode support is used during application development and the version without the debug mode support is used for final testing and distribution.

In the debug mode, the Form Driver rings the terminal bell and uses the bottom screen line to display a message for each of the status conditions (except "Successful completion" and "Current form incomplete") listed in Table 5-1. Appendix E lists the messages. After displaying a debug mode message, the Form Driver places the cursor in the lower right corner of the screen, where it remains until you press the ENTER or RETURN key. This process prevents your program from clearing or overwriting a debug mode message before you have seen it. After you press the ENTER or RETURN key, your program resumes and can then use the FPUTL call to display program-related messages on the bottom screen line.

Except for status code FE$DLN, value −16 (data specified for output too long), the appropriate error code is returned to the calling program.

5.1.3 The Debug Mode and Application Programming Techniques

Because the Form Driver explicitly signals all errors with calls in the debug mode, you can use the Form Driver to debug your program (with respect to Form Driver mistakes). Therefore, after debugging a program, you might choose not to test for certain errors that should not occur in a fully debugged application, such as an incorrect field name or form name, or an incorrect number of arguments in a call.

Some errors can occur even in a fully debugged program. In particular, even in a finished FMS program, you should check for I/O errors after calls that:

- Open and close a form library file.
- Display a form (and must therefore read a form library file).
- Solicit operator responses.

5.1.4 Signaling the Application Operator About Program Errors

In many cases, FMS application operators cannot be expected to learn much about the hardware or software they use. With respect to error conditions that are not totally avoidable, the application designer and programmer are under pressure to signal the problem and its solution as clearly as possible. The FPUTL call is especially useful for giving operators the messages about the application program status they will find most useful. For example, the following general illustration shows one way the FPUTL call can be used with the other status and error-checking features:

1. The program encounters an I/O error while trying to display a form.
2. The program detects the error by using the FSTAT call. The call returns the error code −18 for an error in reading a form library file.
3. The program uses the status code as an index into a list of program-specific messages and finds the following message: "Proper form not available. Contact Ms. Jackson."
4. The program uses the FPUTL call to display the message on the bottom screen line. Immediately thereafter, the program uses a call for an operator response (the FGET call) to ensure that the message remains visible until the operator sees and responds to it.

Chapter 6 provides a full description of the FPUTL call.

5.2 The Role of Field Terminators

The field terminators define one of the following conditions:

1. The operator wants to work on the next form.
2. The operator wants to work on a different field from the current field.

Each key listed in Table 5-2 controls a field terminator. The Autotab field attribute also controls a unique terminator. When an operator presses a key or completes a field with the Autotab attribute, the Form Driver either processes the terminator itself and displays the effect for the operator or returns a unique field terminator code to your program and leaves the choice of processes to the program. Table 5-2 also lists the process and code the Form Driver uses for each field terminator key.

When you set the VT200 keypad to the alternate keypad mode, the Form Driver also treats the keypad's numeric keys, comma (,) key, hyphen (-) key, and decimal point, or period (.) key as field terminators. The codes for these alternate keypad mode terminators are always returned to your program immediately.

This section describes how your program can use field terminators and Form Driver calls to guide an operator in any order through the fields in a form.

5.2.1 Relationship Between Field Terminators and Form Driver Calls

In effect, the Form Driver works between the operator and the application program the operator is using. When the program initiates a call to get an operator response, the Form Driver allows the operator to type an entry in a field. When the operator presses a field terminator key that completes the call, the Form Driver passes the response and the field terminator code to the program and prohibits the operator from typing anything else.

Only the following four Form Driver calls allow the operator to respond:

1. FGET, to get the value for a specified field and the field terminator.
2. FGETAF, to get the value, field name, or field terminator for any field the operator chooses.
3. FGETAL, to get a concatenated string of all field values for the current form and the last field terminator used.
4. FINLN, to get a concatenated string of the field values from the current line of the specified scrolled area to the last field terminator used.

For each of these four calls, the Form Driver validates all field terminators. For example, with the cursor in the first field in a form, the Form Driver accepts the field terminator for the TAB key but does not accept the field terminator for the BACKSPACE key.

Table 5-3 lists the four calls and the field terminator keys that complete each call. The FGET call gives the program total control over responding to any field terminator. The FGETAF call allows the operator to choose one field but returns control to the program as soon as the operator completes an Autotab field or modifies a field and presses any field terminator key. The FGETAL call leaves the Form Driver in control of responding to any field terminator except when the operator presses the ENTER or RETURN key. The FINLN call leaves the Form Driver in control within a line of a scrolled area.

For a general illustration of the flexibility you get from the set of field terminator features and related calls, compare the following two methods for getting all the current field values from the operator. (The illustration assumes none of the fields has any special attributes, such as the Response Required attribute.)

1. Using the FGETAL call.
   - The program initiates the FGETAL call.
   - The operator uses the field terminator keys that move the cursor from field to field at any time. The Form Driver processes these field terminators without returning them to the program.
   - When the operator presses the ENTER or RETURN key, the Form Driver returns the field terminator code and the string of field values to the program.
   - The program is then in control of what the operator does next.
2. Using a series of FGET calls.
   - The program initiates the FGET call. The operator can only type and change the entry in the specified field.
   - When the operator presses any field terminator key, the Form Driver returns the field terminator code and the single field value to the program. The program is then in control of what the operator does next. For example, on the basis of the field value or the field terminator, the program can specify the same field or another field in the next FGET call.

Table 5-2. Field Terminator Keys, Codes, and Typical Effects

Key	Code		Usage or Meaning
	High-Level Languages (Decimal)	**MACRO-11 (Global)**	
ENTER or RETURN	0.	FT$NTR	Terminates all entries in the form. If the call being processed is an FGETAL, and required entries are not complete, the Form Driver refuses to accept the terminator and the operator remains in control. If required entries are complete, the terminator is always returned to the program. Therefore, the final effect depends on the next call the program initiates for an operator response. If any other call is being processed, only the requirements for the current field must be satisfied. If so, control is returned to the program.
TAB	1.	FT$NXT	Valid only when the current field is not the last field in the form that is not display-only. Moves the cursor to the initial position of the next field. Processed by the Form Driver for the FGETAL and FINLN calls and, until an entry is typed or modified, for the FGETAF call. Returned to the program for the FGET call and, after an entry is typed or modified, the FGETAF call.
	6.	FT$SNX	Scroll forward to the next field. The TAB key terminated input in last field of a scrolled line. Always returned to the program.
BACKSPACE	2.	FT$PRV	Valid only when the current field is not the first field in the form that is not display-only. Moves the cursor to the initial position of the previous field. Processed as for the TAB key.
	7.	FT$SPR	Scroll backward to the previous field. The BACKSPACE key terminated input in the first field in a scrolled line. Always returned to the program.
None (Autotab)	3.	FT$ATB	Processed as for the TAB key.
PF3 (Exit Scrolled Area Backward)	4.	FT$XBK	Moves the cursor out of the scrolled area to the initial position of the previous field the operator is allowed to complete. (Valid input only when the current field is in a scrolled area.)
PF4 (Exit Scrolled Area Forward)	5.		(Valid input only when the current field is in a scrolled area). Moves the cursor out of the scrolled area to the initial position of the next field the operator is allowed to complete.

Table 5-2. Field Terminator Keys, Codes, and Typical Effects (Cont.)

Key	Code		Usage or Meaning
DOWNARROW (Scroll Forward)	8.	FT$SFW	(Valid input only when the current field is in a scrolled area). The scrolled area is scrolled up and the current line remains the same physical line (with new data) or the cursor moves down one line and that line becomes the new current line. The cursor moves to the initial position of the first field the operator is allowed to complete in the current line.
UPARROW (Scroll Backward)	9.	FT$SBK	(Valid input only when the current field is in a scrolled area). The scrolled area is scrolled down and the current line remains the same physical line (with new data) or the cursor moves up one line and that line becomes the new current line. The cursor moves to the initial position of the first field the operator is allowed to complete in the current line.

Table 5-3. The Relationship Between the Calls to Get Operator Responses and the Field Terminators

Call	Field Terminator Keys that Complete the Call
FGET	Any valid field terminator key or the Autotab code.
FGETAF	ENTER, RETURN, or any typed field entry followed by any valid field terminator key or the Autotab code.
FGETAL	ENTER or RETURN.
FINLN	Any valid field terminator key or the Autotab code.

In terms of designing forms and programs for FMS applications, the following principles provide a useful summary of Tables 5-2 and 5-3:

1. Except for the ENTER, RETURN, PF3, and PF4 keys, the effects of the field terminator keys cannot be changed from what DIGITAL has designed in the following cases:
   - For the FGETAL call.
   - For the FINLN call.
   - For the FGETAF call before the operator makes a field entry.
2. When the operator uses the ENTER, RETURN, PF3, or PF4 key, or, in response to the FGET call, any field terminator key, the program alone controls the results the operator sees.

If you use the FGETAL call in a program, the TAB key will always advance the cursor from field to field according to the default order DIGITAL has implemented. However, if you use a series of FGET calls instead of the FGETAL call, the program is passed the field terminator code for the TAB key and can react to it in any way you specify.

For example, you can use the FPFT call. After the operator uses any field terminator that returns control to the application program, the program can initiate the FPFT call, making the Form Driver display the effects of any field terminator key. In the example of an FGET call terminated by pressing the TAB key, the program can react by specifying the BACKSPACE key code in the FPFT call. Then, the effect of the next FGET call would be to move the cursor back to the previous field in the form. Or, you can use another FGET call. Again in the example of an FGET call terminated by pressing the TAB key, the program can react with another FGET call that specifies by name the next field the operator is to complete, regardless of where the field appears on the operator's screen.

5.2.2 Using the Alternate Keypad Mode Terminators

Normally, the numeric and punctuation keys on the VT200 keypad produce the same numbers and characters that the corresponding keyboard keys produce. Therefore, for many common applications the operator can enter numeric data by using the keypad rather than the more cumbersome keyboard arrangement.

For special applications, you can set the VT200 to the alternate keypad mode from your program and then design the applications to use the numeric and punctuation keys on the keypad as field terminator keys. In this case, the Form Driver always passes the alternate keypad mode terminators to the program immediately, regardless of whether the Input Required and Must Fill requirements are satisfied for the form. The *VT200 User Guide* describes how to set the alternate keypad mode. Table 5-4 lists the keypad keys affected by the alternate keypad setting and the code returned to your program for each key.

In each case, the character returned is the last character in the escape sequence generated by the key in alternate keypad mode.

Table 5-4. **Alternate Keypad Mode Field Terminator Keys and Codes for the VT200**

Keypad or Function Key	Symbol	Code Returned Value (Decimal)
Comma (,)	FT$KPC	108.
Hyphen (-)	FT$KPH	109.
Decimal (.)	FT$KDP	110.
0	FT$KP0	112.
1	FT$KP1	113.
2	FT$KP2	114.
3	FT$KP3	115.
4	FT$KP4	116.
5	FT$KP5	117.
6	FT$KP6	118.
7	FT$KP7	119.
8	FT$KP8	120.
9	FT$KP9	121.
F06	FT$F06	49.
F07	FT$F07	50.
F08	FT$F08	51.
F09	FT$F09	52.
F10	FT$F10	53.
F11	FT$F11	55.
F12	FT$F12	56.
F13	FT$F13	57.
F14	FT$F14	58.
F15	FT$F15	60.
F16	FT$F16	61.
F17	FT$F17	63.
F18	FT$F18	64.
F19	FT$F19	65.
F20	FT$F20	66.
Find	FT$FND	33.
Insert Here	FT$INS	34.
Remove	FT$RMV	35.
Select	FT$SEL	36.
Prv Screen	FT$PRS	37.
Nxt Screen	FT$NXS	38.

5.3 The Impure Area

The size of the impure area must satisfy the requirements of the largest form used. The actual size also depends on the programming language you are using for your application. For any high-level language, create an impure area 64 bytes (decimal) larger than the impure area size reported by the Form Editor or Form Utility. For MACRO-11, you do not need to add the 64 bytes (decimal).

NOTE

> Because of operating system factors, a form description on RSX-11M and RSX-11M-PLUS systems requires an impure area 44 bytes (decimal) larger than on RT-11 systems, regardless of the language used for the application. On RSX-11M and RSX-11M-PLUS systems, the Form Editor and Form Utility include this extra requirement when they report the impure area size. On RT-11 systems, they do not include the extra requirement.

The impure area is used by the Form Driver to maintain terminal context between calls. If you issue direct calls from your program to display data on or solicit input from the terminal, rather than using the Form Driver for all terminal I/O, the results of the next Form Driver call might not be as expected.

5.4 Task-Building Programs with Memory-Resident Forms

Memory-resident forms are easily created and included in your FMS application. Use the Form Utility to create object modules of the forms you wish to include in your program, as in the following example:

```
FUT> FORMS.OBJ=DEMLIB.FLB/OB
DM0:[30,10]DEMLIB.FLB;1 Form name? PARTS RET
DM0:[30,10]DEMLIB.FLB;1 Form name? FIRST RET
DM0:[30,10]DEMLIB.FLB; Form name? RET
```

Both form descriptions are included in the module FORMS.OBJ. If you wish, you can put forms in separate files.

When your application is task-built with memory-resident forms, the FDVDAT module of the Form Driver must be referenced before any of the memory-resident forms. This is done by including the Form Driver library with an explicit reference to the FDVDAT in the Task Builder command sequence before any of the files containing forms. The following example illustrates this:

```
TKB> FORDEM,FORDEM/-SP=FORDEM
TKB> HLLFOR,FDVLIB/LB:FDVDAT,FDVLIB/LB
TKB> FORMS
TKB> LB:[1,1]FOROTS/LBFDVLIB/LB:FDVDAT,LB:FDVLIB/LB
TKB> //
```

5.5 FCS and RMS System Support

Two versions of the Form Driver are supplied as object libraries:

- FDVLIB.OLB – the Form Driver library for FCS I/O support.
- FDVLRM.OLB – the Form Driver library for RMS I/O support.

The choice you make depends on the programming language you use for your application as well as the overall design of the application. In regard to the design, base your choice on your experience with and knowledge of the FCS and RMS systems. In regard to the relationship between the programming languages and the two systems, Table 5-5 summarizes the Form Driver requirements.

Table 5-5. FCS and RMS System Requirements for the Form Driver

Language	Requirements for FCS Support	Requirements for RMS Support
BASIC-PLUS-2	None	Required
COBOL-11	None	Required
COBOL-81	None	Required
DIBOL-83	None	Required
FORTRAN IV	Required	None
FORTRAN-77	Optional	Optional
MACRO-11	Optional	Optional

5.6 Using the Form Driver as a Resident Library with FCS Support

The command file FDVRES.CMD builds the Form Driver as a resident library with FCS support. This command file can be modified to include additional routines, such as a high-level language interface, in the library.

5.6.1 Procedure for RSX-11M and RSX-11M-PLUS Systems

To use the Form Driver as a resident library, you must allocate a partition named FDVRES and then install the resident library when you build your application system.

The following command to VMR allocates the partition:

```
SET /MAIN=FDVRES:*:400:COM
```

The partition size (400) must be adjusted for any additional modules added to the Form Driver.

The base address of the partition (*) is specified here in the format of RSX V3.2 VMR. As a command to MCR, the address must be specified as a number.

Details on the allocation of partitions can be found in the operator's procedures manual under the SET command and in the SYSGEN manual in the section describing VMR.

When you build your application system, a privileged user can use the following MCR command to install the resident library from UIC [30,10]:

```
INSTALL [30,10]FDVRES
```

To replace a resident library, install a new copy.

The Form Driver cannot be built as a resident library with RMS support.

The following is an example of a Task Builder command procedure to build a program with resident library support:

```
TASK,TASK=TASK,[30,10]FDVDRS
/
RESLIB=[30,10]FDVRES/RO
PAR=FDVRES:100000:0
//
```

The base of the PAR option must agree with the base given in the PAR option in the FDVTKB command file. The Form Driver is not position independent code (PIC). See Chapter 7 of the *RSX-11M Task Builder Reference Manual* for more details on building and using resident libraries.

5.7 The High-Level Language Interface

A special component of the Form Driver, called the high-level language interface, processes your high-level language Form Driver calls. The interface passes the values you supply to the Form Driver and returns values to your program from the Form Driver.

The high-level language interface is entirely transparent to you and to your program except when you build your FMS application. To use forms, you need only the Form Driver calls. However, as part of the procedure for building a running application, you must link the proper high-level language interface component with the Form Driver and your program. The details for building applications in each language are described later in this chapter.

Most of the mutual requirements for the Form Driver and each high-level language are the same. They are grouped in the following four categories and described in the sections that follow:

1. The input and output arguments for the Form Driver calls.
2. The syntax of the calls and conventions used in this manual to define the syntax for the different languages.
3. The completion status of calls for success and failure.
4. Interpretation of the field terminators that an operator uses while working with your application and using the terminators flexibly.

5.7.1 General Description of the Arguments

Collectively, high-level language calls use arguments to pass values to the Form Driver and to receive values the Form Driver returns. For each call, this manual uses the term *Input Arguments* (or *Inputs*) to refer to the arguments that pass values from your program to the Form Driver. The term *Output Arguments* (or *Outputs*) refers to the arguments for values the Form Driver returns to your program. For example, the FGET call allows an operator to enter data in a field and then returns the field value to the program when the operator finishes. The input arguments for the FGET call are the field name and, if the field is indexed, the field index. The output arguments for the FGET call are the field value when the operator terminated the field and the code for the field terminator.

Table 5-6 shows the abbreviations this manual uses for all the Form Driver call arguments and describes briefly the requirements or value for each input argument and output argument. (In Section 5.11, Table 5-16 is a similar list for MACRO-11.)

The full descriptions of the Form Driver calls in Chapter 6 also use the abbreviations that appear in Table 5-6.

Table 5-6. Summary of Form Driver Inputs and Outputs in High-Level Language Calls

High-Level Language Argument Abbreviation	Requirement or Value
Inputs	
CHAN	A channel number for a form library file.
FID	A field name or a named data label, six characters long, including padding (for FORTRAN IV and FORTRAN-77, add a NULL byte also). To specify a scrolled area, use the name of any field in the scrolled area.
FIDX	A field index for the specified field (when the field is indexed) or the index for a named data value. The argument is ignored unless the Form Driver is processing an indexed field or accessing named data by index.
FLNM	A form library file specification.
FNAME	A form name, six characters long, including padding (for FORTRAN IV and FORTRAN-77, add a NULL byte also).
FVAL	As an input value, the single value or the concatenated values to be displayed: • in a field. • in the top, bottom, or current line of a scrolled area. • in the last line of the screen. • in an entire form.
IMPURE	The name of a subscripted variable (or array) of bytes for the impure area.
LINE	The explicit starting line number for the form, overriding the line number assigned with the Form Editor.
SIZE	The size of the impure area in bytes.
TERM	As an input value, the numeric code for the terminator that the Form Driver is to process.
Outputs	
	The status code is set for all calls.
FID	The current field name or a named data label.
FIDX	A field index.
FLEN	The length of a specified field (not the length of the data the field contains).
FVAL	A named data value, a single field value, or a concatenated string that is composed of several field values (including padding when a value is shorter than its field).

Table 5-6. Summary of Form Driver Inputs and Outputs in High-Level Language Calls (Cont.)

High-Level Language	Requirement or Value
TERM	The numeric code for the key that the operator used to terminate input: • in a field. • in a line of a scrolled area. • in an entire form.
STATUS	A numeric code for the completion status of the last call that was executed.
STAT2	A numeric RMS or FCS status code for detailed information when the STATUS value is –4 or –18.

As shown in Table 5-6, the maximum length of form names and field names is six characters. For FMS applications in the high-level languages except FORTRAN IV, the Form Driver pads form names and field names shorter than six characters with spaces. Form names and field names longer than six characters are truncated when passed to the Form Driver. Field names returned by the Form Driver are six characters long, including any spaces that have been added.

5.7.1.1 Argument Data Types — The data types of the Form Driver arguments depend on the language you are using. Therefore, specific requirements are listed later in this chapter in the sections that provide information specific to the languages.

The general data types the Form Driver uses regularly are integers and alphanumeric strings. Examples of arguments that pass integer values to and from the Form Driver are the arguments for:

- The starting line number for a form.
- The code for the key that an operator uses to finish a field.
- The size of the impure area.

Examples of arguments that pass alphanumeric strings to and from the Form Driver are the arguments for:

- The name of a field.
- A named data value.
- A value to be displayed in a field.

5.7.1.2 Relationship Between Field Lengths and Values — Regardless of the practical purposes of the fields in a form, the Form Driver always treats field values as strings. For example, when the Form Driver returns a field value to your program as an argument to the FGET call, your program receives a string of characters exactly as long as the field you specified. (Except for FORTRAN IV and FORTRAN-77, in which case the data is one byte longer because it is terminated with a null byte.) If the value is shorter than the field, Fill Characters (either zeroes or spaces, as assigned with the Form Editor) are added.

As another example of the relationship between field lengths and values, when you use the FPUTAL call to display specified values in the first three fields of a form, a concatenated string of the three field values passes to the Form Driver. For any value shorter than the field in which it is to be displayed, you add enough Fill Characters so that the value and the field are the same length.

5.7.2 General Description of Call Syntax for High-Level Languages

The syntax of the Form Driver calls follow the requirements and conventions of the language you use. BASIC-PLUS-2, COBOL-11, COBOL-81, FORTRAN IV, and FORTRAN-77 calls all use the CALL statement. DIBOL-83 uses the XCALL statement. In this manual, only the CALL statement forms are listed in the detailed descriptions of the calls. For example, the CALL statement syntax for the calls to get a field value from the operator and then display a message on the last line of the operator's screen is as follows:

1. For BASIC-PLUS-2, FORTRAN IV, and FORTRAN-77

   `CALL FGET`*(fval,term,fid[,fidx])*

   `CALL FPUTL`*(fval)*

2. For COBOL-11 and COBOL-81

   CALL "FGET" USING BY DESCRIPTOR *fval,*
   BY REFERENCE *term,*
   BY DESCRIPTOR *fid*
   [,BY REFERENCE *fidx*]

   CALL "FPUTL" USING BY DESCRIPTOR *fval*

3. For DIBOL-83

   XCALL FGET*(fval,term,fid[,fidx])*

   XCALL FPUT*(fval)*

The argument abbreviations printed in lowercase letters stand for arguments that you must provide. They must be in the order shown for each call and must meet the functional requirements described in Table 5-6.

In the call descriptions in this manual, square brackets ([and]) enclose optional arguments. For example, in the FGET call illustrated above, the argument for a field index (*fidx*) is required only to specify a particular field in an indexed field.

For calls with more than one optional field, omitting one requires omission of any others to its right. For example, if you omit the optional field name argument (*fid*) in the following call, you must also omit the field value argument (*fval*).

CALL FPFT*(term[,fid[,fval]])*

CALL "FPFT" USING *term*
[,BY DESCRIPTOR *fid[,fval]]*

For some calls, you can omit the entire list of arguments. For added clarity in this manual, these calls are listed both with and without the argument lists. For example, the full syntax of the FPUTAL call is shown as follows:

CALL FPUTAL*(fval)*

CALL FPUTAL

The Form Driver high-level language interface does not support null argument lists in calls. Using the second form of the FPUTAL call above as an example, the following format is invalid at all times within an FMS application:

```
CALL FPUTAL()                 !The null argument form is
                              always invalid with FMS.
```

With FORTRAN IV and FORTRAN-77, you can also use the standard syntax for calling a function subprogram. The function subprogram syntax for the FGET and FPUTL calls is:

fncval = FGET(*fval,term,fid[,fidx]*)

fncval = FPUTL(*fval*)

The argument fncval stands for the value of the function.

When you use a Form Driver call as a function, the value of the function is the Form Driver completion status for the call. The next section explains status and error checking in greater detail.

5.7.3 Status and Error Checking

As described in Section 6.25, the Form Driver includes a specific call, the FSTAT call, that returns status codes for the completion status of the last call processed. Table 5-1 lists the status codes and their meanings.

5.8 The Interface for BASIC-PLUS-2

In BASIC-PLUS-2 applications, all numeric values passed to and from the Form Driver must be integer variables or constants. String values must also be string variables or constants. When the Form Driver returns a string value, the length is the length of the field, including any trailing spaces or fill characters. String values that are shorter than the BASIC-PLUS-2 variables to which they are assigned are left-justified and the fields are blank-filled. String values that are longer than the variables to which they are assigned are truncated and cause the Form Driver to set the status code to −22.

BASIC-PLUS-2 programs should use RMS support.

To avoid loss of typeahead on RSX-11M and RSX-11M-PLUS systems, FMS applications must attach the terminal. This is done by calling the subroutine "WTQIO," which is the queue I/O request and wait call (see the description of QIOW$ in the *RSX-11M Executive Manual*). The arguments are the same as for the FORTRAN form of the call: INUM is 768 and LUN is the LUN assigned to the terminal for the Form Driver, as shown in the following example:

```
CALL WTQIO (768%,5%,5%)
```

5.8.1 Arguments for the Calls

Table 5-7 lists typical BASIC-PLUS-2 data types and data structures for each of the arguments in the Form Driver calls.

Table 5-7. Typical BASIC-PLUS-2 Data Types for Form Driver Arguments

Argument Abbreviation	Purpose, Data Type, and Data Structure
CHAN	Channel number: integer variable or constant.
FID	Field name: 6-byte string variable or constant.
FIDX	Field and named data index: integer variable or constant.
FLEN	Field length: integer variable or constant.
FLNM	Form library file specification: string variable or constant. (The size depends on application requirements and conventions.)
FNAME	Form name: 6-byte string variable or constant.
FVAL	Named data value, one or more field values, text for display on the bottom screen line: string variable or constant. (The size depends on the application.)
IMPURE	Impure area: byte array (using the impure area size that the Form Editor and Form Utility report, the size of the array should be 64. bytes larger than the largest impure area for the forms that the application uses).
LINE	Starting line number for a displayed form: integer variable or constant.
SIZE	The size of the impure area in bytes.
STATUS	Call completion status: integer variable.
STAT2	FCS or RMS system error code: integer variable.
TERM	Field terminator code: integer variable or constant.

5.8.2 Syntax for the Calls

All of the Form Driver calls use the CALL statement. Table 5-8 contains a summary of the function and the full CALL statement syntax for each call. Lowercase letters represent the arguments you must supply; optional arguments are enclosed in square brackets ([and]). Calls that need no arguments are listed separately. Argument abbreviations and functions are described in Table 5-7.

Table 5-8. Listing of BASIC-PLUS-2 Form Driver Calls

Call Abbreviation	Summary and Forms
FCHIMP	For high-level languages only, switches control from one impure area to another. The form is: `CALL FCHIMP`*(impure)*
FCLRSH	Clears the entire screen and displays the form with the default field values. If a line number is specified, uses it as the starting line number for the form. The form is: `CALL FCLRSH`*(fnam[,line])*
FGCF	Returns the field name from the Form Driver Argument List (and if it is an indexed field, its index). The form is: `CALL FGCF`*(fid[,fidx])*
FGET	If a field name is specified, gets and returns the value for the field and the field terminator used. If no field name is specified, places the cursor at the lower right corner of the screen and deactivates all operator responses except the RETURN and ENTER keys. The forms are: `CALL FGET`*(fval,term,fid[,fidx])* `CALL FGET`
FGETAF	Gets and returns the value, field name (and, if the field is indexed, its index) and the field terminator used for the field that the operator chooses. The form is: `CALL FGETAF`*(fval,term,fid[,fidx])*
FGETAL	If the call includes an argument, gets and returns a concatenated string of all field values (and optionally the last field terminator used). If no arguments are specified, gets all values from the operator but only stores them in the impure area. The forms are: `CALL FGETAL`*(fval[,term])* `CALL FGETAL`
FIDATA	Gets and returns the named data value that has the specified index. The form is: `CALL FIDATA`*(fidx,fval[,fid])*
FINIT	Initializes the impure area for high-level languages and supplies the name and size of that impure area to the Form Driver. The form is: `CALL FINIT`*(impure,size[,status])*
FINLN	Gets and returns a concatenated string of the field values for the current line of the scrolled area that contains the specified field name and the last terminator used. The form is: `CALL FINLN`*(fid,fval,term)*
FLCHAN	Supplies to the Form Driver the I/O channel (LUN) to use for reading a form library file. The form is: `CALL FLCHAN`*(chan)*
FLCLOS	Closes the current form library file. The form is: `CALL FLCLOS`

Table 5-8. **Listing of BASIC-PLUS-2 Form Driver Calls (Cont.)**

Call Abbreviation	Summary and Forms
FLEN	Returns the length of the specified field. The form is: `CALL FLEN`*(flen,fid[,fidx])*
FLOPEN	Opens the specified form library file. The form is: `CALL FLOPEN`*(flnm)*
FNDATA	Gets and returns the named data value that has the named data label specified. The form is: `CALL FNDATA`*(fid,fval)*
FOUTLN	Displays the specified string of field values in the current line of the scrolled area that contains the specified field. The form is: `CALL FOUTLN`*(fid,fval)*
FPFT	If the call includes an argument, processes the specified field terminator and identifies the appropriate field as the current field. (To get the name of the field, use the FGCF call.) If the specified terminator is a scrolled area terminator, the name of a field in the intended scrolled area must be specified, and if a string of values is specified, the values will be displayed on the top or bottom line of the scrolled area after the terminator is processed. If no argument is included, the call processes the last terminator that was used. The forms are: `CALL FPFT`*(term[,fid[,fval]])* `CALL FPFT`
FPUT	Displays the specified value in the specified field. The form is: `CALL FPUT`*(fval,fid[,fidx])*
FPUTAL	Displays values in all fields of the form. If a concatenated string of values is supplied, each value must be the same length as the field in which it is to be displayed and the values must be in the same order that the FGETAL call would produce for the form. Values from the supplied string are displayed in the first fields of the form, and defaults are displayed in any fields that remain. If no string of values is supplied, default values are displayed in all fields. The forms are: `CALL FPUTAL`*(fval)* `CALL FPUTAL`
FPUTL	If an argument is specified, displays the specified string on the bottom line of the screen. If no argument is specified, clears the bottom line. The forms are: `CALL FPUTL`(fval) `CALL FPUTL`
FRETAL	Returns the current values for all fields in the form in the same order that the FGETAL call would produce. The form is: `CALL FRETAL`*(fval)*
FRETN	Returns the current value of the specified field. The form is: `CALL FRETN`*(fval,fid[,fidx])*

Table 5-8. Listing of BASIC-PLUS-2 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FSHOW	Clears the part of the screen that the specified form requires and displays the form with default field values. If a line number is specified, uses it as the starting line number for the form. The form is: `CALL FSHOW`*(fnam[,line])*
FSPOFF	Turns off the supervisor-only mode and allows the operator to enter and change data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The form is: `CALL FSPOFF`
FSPON	Turns on the supervisor-only mode and prevents the operator from entering or changing data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The form is: `CALL FSPON`
FSTAT	Returns the status code for the last call that was processed as the value of the first argument. The value of the second argument is meaningful as an FCS or RMS system error code (depending on the version of the Form Driver in use) only if the value of the first argument is –4 or –18, indicating an error while trying to open or read a form library file. The form is: `CALL FSTAT`*(status[,stat2])*

5.8.3 Building a BASIC-PLUS-2 Task

Building BASIC-PLUS-2 tasks involves editing the command and ODL files produced by BASIC-PLUS-2.

Appendix D contains source listings of command files used to task-build FMS BASIC-PLUS-2 sample application programs.

5.9 The Interface for COBOL-11 and COBOL-81

In COBOL programs, only data names can be passed as arguments, except in the cases of FPUTL and FLOPEN. All values that are passed must have been defined in the data division of the program. No string literals or numeric constants are allowed.

The COBOL interface assumes two specific data types when passing data to and from the Form Driver. These data types are:

for strings: left-justified sign separate

for numbers: computational

See Table 5-9 for a list of COBOL arguments showing the necessary data types. You can create your own data structure provided you use the listed data types in your Form Driver calls.

COBOL tasks should use the Form Driver with RMS support.

To avoid loss of typeahead on RSX-11M and RSX-11M-PLUS systems, FMS applications must attach the terminal. This is done by calling the subroutine "WTQIO," which is the queue I/O request and wait call (see the description of QIOW$ in the *RSX-11M Executive Manual*). The arguments are the same as for the FORTRAN form of the call: INUM is 768 and LUN is the LUN assigned to the terminal for the Form Driver. For example:

```
INUM        PIC 999    comp value 768.
LUN         PIC 9      comp value 1.
.
.
.
CALL "WTQIO" USING INUM,LUN,LUN.
```

5.9.1 Using the Form Utility (FUT) to Create theCommunication Structure for a COBOL Program

The Form Utility (FUT) creates the communication structure for a form used by a COBOL program. (See Chapter 3 on the Form Utility.) In response to the /CC option, FUT creates a COBOL library file. The output is a text file with the default file type .LIB.

At compile time, you request the library file with a COPY command in the data division of your program. (See the *PDP-11 COBOL Language Reference Manual* for details on the COPY command.)

Group items created by FUT have the same names as the field names in the form you are using.

The library file contains the necessary communication structure. If you do not wish to use the structure provided, you can create your own.

The output of the Form Utility is in terminal format with respect to the COBOL program. If you want to use conventional format, you must reformat the file. (See the Reformat utility in the *PDP-11 COBOL User's Guide*. On COBOL program formats, see the *PDP-11 COBOL Language Reference Manual*.)

Chapter 3 includes an example of the COBOL structure that the Form Utility can produce.

5.9.2 Arguments for the Calls

Table 5-9 lists typical PDP-11 COBOL data types and data structures for each of the arguments in the Form Driver calls.

Table 5-9. Typical COBOL-11 and COBOL-81 Data Types for Form Driver Arguments

Argument Abbreviation	Purpose, Data Type, and Picture Attributes
CHAN	Channel number: binary index Picture: one-word computational, synchronized left
FID	Field name: 6-character string Picture: any character, blank padded, left-justified, sign separate, synchronized left
FIDX	Field and named data index: binary index Picture: one-word computational, synchronized left
FLEN	Field length: binary index Picture: one-word computational, synchronized left
FLNM	Form library file specification: a string whose length is the length of the file specification plus two characters for the delimiters that enclose the specification. Picture: any character, blank padded, left-justified, sign separate, synchronized left
FNAME	Form name: 6-character string. Picture: any character, blank padded, left-justified, sign separate, synchronized left
FVAL	Named data value, one or more field values, text for display on the bottom screen line. A string whose length is: • For a single field value, the length of the field. • For a string of field values, the sum of the lengths of all fields to be processed. • For a named data value, the length of the named data field maximum 60– actual length is variable. • For text to be displayed on the bottom line, the length of the text plus 2 characters for the delimiters that enclose the text.
IMPURE	Impure area: (using the impure area size that the Form Editor and Form Utility report, the size of the variable should be 64 bytes (decimal) larger than the largest impure area for the forms that the application uses).

Table 5-9. Typical COBOL-11 and COBOL-81 Data Types for Form Driver Arguments (Cont.)

Argument	Purpose, Data Type, and Picture Attributes
LINE	Starting line number for a displayed form: binary index. Picture: one-word computational, synchronized left.
SIZE	The size of the impure area in bytes.
STATUS	Call completion status: binary index. Picture: one-word signed computational, synchronized left.
STAT2	FCS or RMS system error code: binary index. Picture: one-word signed computational, synchronized left.
TERM	Field terminator code: binary index. Picture: one-word (two bytes) computational, synchronized left.

5.9.3 Syntax for the Calls

All of the Form Driver calls use the CALL statement. Table 5-10 summarizes the principal purposes and shows the full CALL statement syntax for each call. The arguments you must supply are in lowercase letters, and optional arguments are enclosed in square brackets ([and]). The forms of calls that have no arguments are listed separately. The argument abbreviations and purposes are fully described in Table 5-10.

Table 5-10. Listing of COBOL-11 and COBOL-81 Form Driver Calls

Call Abbreviation	Summary and Forms
FCHIMP	For high-level languages only, switches control from one impure area to another. The form is: `CALL "FCHIMP" USING BY DESCRIPTOR` *impure*
FCLRSH	Clears the entire screen and displays the form with default field values. If a line number is specified, uses it as the starting line number for the form. The form is: `CALL "FCLRSH" USING BY DESCRIPTOR` *fnam* `[,BY REFERENCE` *line*`]`
FGCF	Returns the field name from the Form Driver Argument List (and if it is an indexed field, its index). The form is: `CALL "FGCF" USING BY DESCRIPTOR` *fid* `[,BY REFERENCE` *fidx*`]`

Table 5-10. Listing of COBOL-11 and COBOL-81 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms

FGET	If a field name is specified, gets and returns the value for the field and the field terminator used. If no field name is specified, synchronizes the program with the operator.
	
	The forms are: `CALL "GET" USING BY DESCRIPTOR` *fval*
	`BY REFERENCE` *ter,*
	`BY DESCRIPTOR` *fid*
	`[,BY REFERENCE` *fidx*`]`
	`CALL "FGET"`

FGETAF	Gets and returns the value, field name (and if the field is indexed, its index), and the field terminator used for the field that the operator chooses. The form is: `CALL "FGETAF" USING BY DESCRIPTOR` *fval,* `BY REFERENCE` *term,* `BY DESCRIPTOR` *fid* `[,BY REFERENCE` *fidx*`]`

FGETAL	If the call includes an argument, gets and returns a concatenated string of all field values (and optionally the last field terminator used). If no arguments are specified, gets all values but only stores them in the impure area.
	
	The forms are: `CALL "FGETAL" USING BY DESCRIPTOR` *fval*
	`[,BY REFERENCE` *term*`]`
	`CALL "FGETAL"`

FIDATA	Gets and returns the named data value that has the index specified. The form is: `CALL "FIDATA" USING BY REFERENCE` *fidx,* `BY DESCRIPTOR` *fval,* `[BY DESCRIPTOR` *fid*`].`
FINIT	Supplies to the Form Driver the name of the impure area to use and its size. This call returns its own status code if the third argument is specified. The form is: `CALL "FINIT" USING BY DESCRIPTOR` *impure,* `BY REFERENCE` *size[,status].*
FINLN	Gets and returns a concatenated string of the field values for the current line of the scrolled area that contains the specified field name and the last terminator used. The form is: `CALL "FINLN" USING BY DESCRIPTOR` *fid,* `BY DESCRIPTOR` *fval,* `BY REFERENCE` *term.*
FLCHAN	Supplies to the Form Driver the I/O channel (LUN) to use for reading a form library file. The form is: `CALL "FLCHAN" USING BY REFERENCE` *chan.*
FLCLOS	Closes the current form library file. The form is: `CALL "FLCLOS"`

Table 5-10. Listing of COBOL-11 and COBOL-81 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FLEN	Returns the length of the specified field. The form is: `CALL "FLEN" USING BY REFERENCE` *flen,* `BY DESCRIPTOR` *fid* `[,BY REFERENCE` *fidx*`].`
FLOPEN	Opens the specified form library file. The form is: `CALL "FLOPEN" USING BY DESCRIPTOR` *flnm.*
FNDATA	Gets and returns the named data value that has the named data label specified. The form is: `CALL "FNDATA" USING BY DESCRIPTOR` *fid,* `BY DESCRIPTOR` *fval.*
FOUTLN	Displays the specified string of field values in the current line of the scrolled area that contains the specified field. The form is: `CALL "FOUTLN" USING BY DESCRIPTOR` *fid,* `BY DESCRIPTOR` *fval.*
FPFT	If the call includes an argument, processes the specified field terminator and identifies the appropriate field as the current field. (To find the current field name, use the FGCF call.) If the specified terminator is a scrolled area terminator, the name of a field in the intended scrolled area must be specified, and if a string of values is specified, the values will be displayed on the top or bottom line of the scrolled area after the terminator is processed. If no argument is included, the call processes the last terminator that was used. The forms are: `CALL "FPFT" USING BY REFERENCE` *term* `[,BY DESCRIPTOR` *fid[,fval]]*`.` `CALL "FPFT".`
FPUT	Displays the specified value in the specified field. The form is: `CALL "FPUT" USING BY DESCRIPTOR` *fval,* `BY DESCRIPTOR` *fid* `[,BY REFERENCE` *fidx*`].`
FPUTAL	Displays values in all fields of the form. If a concatenated string of values is supplied, each value must be the same length as the field in which it is to be displayed and the values must be in the same order that the FGETAL call would produce for the form. Values from the string supplied are displayed in the first fields of the form, and defaults are displayed in any fields that remain. If no string of values is supplied, default values are displayed in all fields. The forms are: `CALL "FPUTAL" USING BY REFERENCE` *fval.* `CALL "FPUTAL".`
FPUTL	If an argument is specified, displays the specified string on the bottom line of the screen. If no argument is specified, clears the bottom line. The forms are: `CALL "FPUTL" USING BY REFERENCE` *fval.* `CALL "FPUTL".`

Table 5-10. Listing of COBOL-11 and COBOL-81 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FRETAL	Returns the current values for all fields in the form in the same order that the FGETAL call would produce. The form is: `CALL "FRETAL" USING BY REFERENCE` *fval.*
FRETN	Returns the current value of the specified field. The form is: `CALL "FRETN" USING BY REFERENCE` *fval,* `BY REFERENCE` *fid[,fidx].*
FSHOW	Clears the area of the screen specified when the form was created and displays the form with default field values. If a line number is specified, uses it as the first line for the form. The form is: `CALL "FSHOW" USING BY REFERENCE` *fnam[,line].*
FSPOFF	Turns off the supervisor-only mode and allows the operator to enter and change data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The form is: `CALL "FSPOFF".`
FSPON	Turns on the supervisor-only mode and prevents the operator from entering or changing data in fields to which the Supervisor Only attribute was assigned in the Form Editor. The form is: `CALL "FSPON".`
FSTAT	Returns the status code for the last call that was processed as the value of the first argument. The value of the second argument is meaningful as an FCS or RMS system error code (depending on the version of the Form Driver in use) only if the value of the first argument is –4 or –18, indicating an error while trying to open or read a form library file. The form is: `CALL "FSTAT" USING BY REFERENCE` *status[,stat2].*

5.9.4 Building a COBOL Program

Appendix D contains examples of the command files used to build the following COBOL demos:

- C11RMSCLS.CMD, a command file to build a COBOL-11 demo
- C11RMSRES.CMD, a command file to build a COBOL-11 demo with RMSRES resident library
- C11RMSTKB.CMD, a command file to build a COBOL-11 demo
- C81RMSCLS.CMD, a command file to build a COBOL-81 demo
- C81RMSRES.CMD, a command file to build a COBOL-81 demo with RMSRES resident library
- C81RMSTKB.CMD, a command file to build a COBOL-81 demo

For more information refer to the *RSX-11M Task Builder Reference Manual.*

Here is a complete sample session to compile and build the COBOL-81 demo:

```
>C81     C81DEM,C81DEM=COBDEM/CVF/-CIS/ERR:1
>TKB     @C81RMSTKB
>RUN     C81RMSDEM
```

You should use the /ERR:1 compiler option because Form Driver calls allow a variable number of arguments. If you do not use the /ERR:1 switch option, COBOL will produce warnings when you use optional argument lists. However, the /ERR:1 option also suppresses other warning diagnostics.

If you elect to build an overlaid COBOL application, problems may arise during task-building. This can occur if the COBOL MERGE utility is used and the resulting ODL file edited. This is because the Task Builder will not accept files with lines longer than 80 characters (decimal). COBOL MERGE creates lines that are blank filled to exactly 80 characters (decimal). If you use an editor to modify the ODL file, it is possible characters will be inserted and the lines will be longer than the Task Builder's maximum. You can solve this problem by deleting all trailing blanks from any line modified by an editor.

5.10 The Interface for DIBOL-83

In DIBOL-83 applications, all values passed to and from the Form Driver must be ASCII string variables or literals. When the Form Driver returns a string value, the length is the length of the field, including any trailing spaces or fill characters. String values that are shorter than the DIBOL-83 variables to which they are assigned are left-justified and the fields are blank-filled. String values that are longer than the variables to which they are assigned result in DIBOL-83 run-time error message '?ERR 31 ARGUMENT WRONG SIZE' and cause the Form Driver to set the status code to −22.

5.10.1 Arguments for the Calls

Table 5-11 lists typical DIBOL-83 data types and data structures for each of the arguments in the Form Driver calls.

Table 5-11. Typical DIBOL-83 Data Types for Form Driver Arguments

Argument Abbreviation	Purpose, Data Type, and Data Structure
CHAN	Channel number: ASCII numeric string variable or literal
FID	Field name: 6-byte ASCII string variable or literal
FIDX	Field and named data index: ASCII numeric string variable or literal
FLEN	Field length: ASCII numeric variable or literal
FLNM	Form library file specification: ASCII string variable or literal (must incorporate a trailing space)
FNAM	Form name: 6-byte ASCII string variable or literal
FVAL	Named data value, one or more field values, text for display on the bottom screen line: ASCII string variable or literal (the size depends on the application).
TID	Keyboard string that identifies the terminal. The default can be identified by KB:; a specific terminal can be identified by KBn:, where n is the terminal number.
IMPURE	Impure area: byte array (using the impure area size that the Form Editor and the Form Utility report, the size of the array should be 64 bytes larger than the largest impure area for the forms that the application uses).
LINE	First line for a displayed form: ASCII numeric variable or literal
SIZE	The size of the impure area in bytes: ASCII numeric variable
STATUS	Call completion status: ASCII numeric string variable
STAT2	RSTS/E system error code: ASCII numeric string variable
TERM	Field terminator code: ASCII string variable or literal

5.10.2 Syntax for the Calls

All DIBOL-83 Form Driver calls use the XCALL statement. Table 5-12 summarizes the functions and shows the full XCALL statement syntax for each call. The arguments that you must supply are in lowercase letters, and optional arguments are enclosed in square brackets ([and]). The formats of calls that have no arguments are listed separately. (The argument abbreviations and functions are described in Table 5-11.)

Since DIBOL-83 external subroutine names must have five characters or less, all six-character names of Form Driver routines are truncated to five characters by removal of the first character ("F").

The name of the Forms Library opened by FLOPEN (in DIBOL-83, LOPEN) must have a trailing space following the name.

Table 5-12. Listing of DIBOL-83 Form Driver Calls

Call Abbreviation	Summary and Forms
CLRSH	Clears the entire screen and displays the form with the default field values. If a line number is specified, uses it as the first line for the form. The format is: `XCALL CLRSH`*(fnam[,line])*
DETCH	The detach terminal call should be issued by a program prior to exiting to ensure that the terminal is not left in an unusual state (e.g., with video attributes set or a scrolled area other than the entire screen). This call may also be used to allow an application program to issue standard terminal input requests. If the FDETCH call is used to allow standard terminal input, the FATTCH call must be issued to attach the terminal before any other Form Driver calls are issued. The format is: `XCALL DETCH`
FGCF	Returns the field name from the Form Driver Argument List (and, if it is an indexed field, its index). The format is: `XCALL FGCF`*(fid[,fidx])*
FGET	If a field name is specified, gets and returns the value for the field and the field terminator used. If no field name is specified, places the cursor at the lower right corner of the screen and deactivates all operator responses except the RETURN and ENTER keys. The formats are: `XCALL FGET`*(fval,term,fid[,fidx])* `XCALL FGET`
GETAF	Gets and returns the value, field name (and, if the field is indexed, its index), and the field terminator used for the field that the operator chooses. The format is: `XCALL GETAF`*(fval,term,fid[,fidx])*

Table 5-12. Listing of DIBOL-83 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
GETAL	If the call includes an argument, gets and returns a concatenated string of all field values (and optionally the last field terminator used). If no arguments are specified, gets all values from the operator but only stores them in the impure area. The formats are: `XCALL GETAL`*(fval[,term])* `XCALL GETAL`
IDATA	Gets and returns the named data value that has the specified index. The format is: `XCALL IDATA`*(fidx,fval[,fid])*
FINIT	Supplies to the Form Driver the name of the impure area to use and its size. (The 'size' argument need not be passed unless the 'status' argument is also being passed since size is already conveyed by the DIBOL-83 'size, origin descriptor' passed for every argument.) The format is: `XCALL FINIT`*(impure,size[,status])*
FINLN	Gets and returns a concatenated string of the field values for the current line of the scrolled area that contains the specified field name and the last terminator used. The format is: `XCALL FINLN`*(fid,fval,term)*
LCHAN	Supplies to the Form Driver the I/O channel (LUN) to use for reading a form library file. The format is: `XCALL LCHAN`*(chan)*
LCLOS	Closes the current form library file. The format is: `XCALL LCLOS`
FLEN	Returns the length of the specified field. The format is: `XCALL FLEN`*(flen,fid[,fidx])*
LOPEN	Opens the specified form library file. The format is: `XCALL LOPEN`*(flnm)*
NDATA	Gets and returns the named data value that has the specified named data label. The format is: `XCALL NDATA`*(fid,fval)*
OUTLN	Displays the specified string of field values in the current line of the scrolled area that contains the specified field. The format is: `XCALL OUTLN`*(fid,fval)*
FPFT	If the call includes an argument, processes the specified field terminator and identifies the appropriate field as the current field. To get the name of the field, use the FGCF call. If the specified terminator is a scrolled area terminator, the name of a field in the intended scrolled area must be specified, and if a string of values is specified, they will be displayed on the top or bottom line of the scrolled area after the terminator is processed. If no argument is included, the call processes the last terminator that was used. The formats are: `XCALL FPFT`*(term[,fid[,fval]])* `XCALL FPFT`

Table 5-12. Listing of DIBOL-83 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FPUT	Displays the specified value in the specified field. The format is: `XCALL FPUT`*(fval,fid[,fidx])*
PUTAL	Displays values in all fields of the form. If a concatenated string of values is supplied, each value must be the same length as the field in which it is to be displayed and the values must be in the same order that the GETAL call would produce for the form. Values from the supplied string are displayed in the first fields of the form, and defaults are displayed in any fields that remain. If no string of values is supplied, default values are displayed in all fields. The formats are: `XCALL PUTAL`*(fval)* `XCALL PUTAL`
FPUTL	If an argument is specified, displays the specified string on the bottom line of the screen. If no argument is specified, clears the bottom line. The formats are: `XCALL FPUTL`*(fval)* `XCALL FPUTL`
RETAL	Returns the current values for all fields in the form in the same order that the GETAL call would produce. The format is: `XCALL RETAL`*(fval)*
FRETN	Returns the current value of the specified field. The format is: `XCALL FRETN`*(fval,fid[,fidx])*
FSHOW	Clears the area of the screen specified when the form was created and displays the form with default field values. If a line number is specified, uses it as the first line for the form. The format is: `XCALL FSHOW`*(fnam[,line])*
SPOFF	Turns off the supervisor-only mode and allows the operator to enter and change data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The format is: `XCALL SPOFF`
FSPON	Turns on the supervisor-only mode and prevents the operator from entering or changing data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The format is: `XCALL FSPON`
FSTAT	Returns the status code for the last call that was processed as the value of the first argument. The value of the second argument is meaningful as an RMS system error code (depending on the version of the Form Driver in use) only if the value of the first argument is –4 or –18, indicating an error while trying to open or read a form library file. The format is: `XCALL FSTAT`*(status[,stat2])*

5.10.3 Building a DIBOL-83 Task

Appendix D contains source listings of command files used to task-build FMS DIBOL-83 sample application programs.

5.11 The Interface for FORTRAN IV and FORTRAN-77

The calling sequences for FORTRAN IV and FORTRAN-77 are identical.

Numeric arguments must be one-word integers. If you use real numbers of bytes instead, the calls will not work properly.

Strings returned from the Form Driver are ASCIZ, and strings input to the Form Driver must be ASCIZ. "ASCIZ strings" contain a null byte as their last character. Therefore, programs must allocate an extra byte for data returned from the Form Driver.

For literals, enclosed in quotation marks, that are passed as arguments, FORTRAN generates ASCIZ strings. The best way to implement string variables is by byte arrays. The variables can be passed as arguments and manipulated one character at a time. The data in the array must end with a null.

All subroutines can be called either as subprograms or as functions. If they are called as functions, the name of the routine must be declared in an integer statement or an implicit integer statement (for example, IMPLICIT INTEGER (F)). If called as functions, the subroutines return the status of the call from the Form Driver on output.

FORTRAN-77 programs can use the Form Driver with FCS or RMS support.

FORTRAN IV programs should use the Form Driver with FCS support. To avoid loss of typeahead, FMS applications must attach the terminal. The FORTRAN programmer attaches the terminal with the high-level language interface "WTQIO" call, (see the description of QIOW$ in the *RSX-11M Executive Manual*) which is the queue I/O request and wait call. The INUM is 768. LUN is the LUN assigned as terminal for the Form Driver.

```
CALL WTQIO (768,5,5)
```

5.11.1 Arguments for the Calls

Table 5-13 lists typical FORTRAN IV and FORTRAN-77 data types and data structures for each of the arguments in the Form Driver calls.

Table 5-13. Typical FORTRAN IV and FORTRAN-77 Data Types for Form Driver Arguments

Argument Abbreviation	Purpose, Data Type, and Data Structure
CHAN	Channel number: integer variable or constant
FID	Field name: 7-byte string variable
FIDX	Field and named data index: integer variable
FLEN	Field length: integer variable or constant
FLNM	Form library file specification: string or constant. (The size depends on application requirements and conventions.)
FNAME	Form name: 7-byte string variable or constant
FVAL	Named data value, one or more field values, text for display on the bottom screen line: string variable or constant. (The size depends on the application.)
IMPURE	Impure area: byte array (using the impure area size that the Form Editor and Form Utility report, the size of the array should be 64 bytes (decimal) larger than the largest impure area for the forms that the application uses).
LINE	Starting line number for a displayed form: integer variable or constant
SIZE	The size of the impure area array in bytes
STATUS	Call completion status: integer variable
STAT2	FCS or RMS system error code: integer variable
TERM	Field terminator code: integer variable

5.11.2 Syntax for the Calls

All of the Form Driver calls use the CALL statement. Table 5-14 summarizes the principal purposes and shows the full CALL statement syntax for each call. The arguments you must supply are in lowercase letters, and optional arguments are enclosed in square brackets ([and]). The forms of calls that have no arguments are listed separately. The argument abbreviations and purposes are described in Table 5-11.

Table 5-14. Listing of FORTRAN IV and FORTRAN-77 Form Driver Calls

Call Abbreviation	Summary and Forms
FCHIMP	For high-level languages only, switches control from one impure area to another. The form is: `CALL FCHIMP` *(impure)*
FCLRSH	Clears the entire screen and displays the form with the default field values. If a line number is specified, uses it as the starting line number for the form. The form is: `CALL FCLRSH` *(fnam[,line])*
FGCF	Returns the field name from the Form Driver Argument List (and, if it is an indexed field, its index). The form is: `CALL FGCF` *(fid[,fidx])*
FGET	If a field name is specified, gets and returns the value for the field and the field terminator used. If no field name is specified, synchronizes the program with the operator. The forms are: `CALL FGET`*(fval,term,fid[,fidx])* `CALL FGET`
FGETAF	Gets and returns the value, field name (and, if the field is indexed, its index) and the field terminator used for the field that the operator chooses. The form is: `CALL FGETAF`*(fval,term,fid[,fidx])*
FGETAL	If the call includes an argument, gets and returns a concatenated string of all field values (and optionally the last field terminator used). If no arguments are specified, gets all values from the operator but only stores them in the impure area. The forms are: `CALL FGETAL`*(fval[,term])* `CALL FGETAL`
FIDATA	Gets and returns the named data value that has the index specified. The form is: `CALL FIDATA`*(fidx,fval[,fid])*
FINIT	Supplies to the Form Driver the name of the impure area to use, and its size. The form is: `CALL FINIT`*(impure,size[,status])*
FINLN	Gets and returns a concatenated string of the field values for the current line of the scrolled area that contains the specified field name and the last terminator used. The form is: `CALL FINLN`*(fid,fval,term)*
FLCHAN	Supplies to the Form Driver the I/O channel (LUN) to use for reading a form library file. The form is: `CALL FLCHAN`*(chan)*
FLCLOS	Closes the current form library file. The form is: `CALL FLCLOS`

Table 5-14. Listing of FORTRAN IV and FORTRAN-77 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FLEN	Returns the length of the specified field. The form is: `CALL FLEN`*(flen,fid[,fidx])*
FLOPEN	Opens the specified form library file. The form is: `CALL FLOPEN`*(flnm)*
FNDATA	Gets and returns the named data value that has the named data label specified. The form is: `CALL FNDATA`*(fid,fval)*
FOUTLN	Displays the specified string of field values in the current line of the scrolled area that contains the specified field. The form is: `CALL FOUTLN`*(fid,fval)*
FPFT	If the call includes an argument, processes the specified field terminator and identifies the appropriate field as the current field. (To get the name of the field, use the FGCF call.) If the specified terminator is a scrolled area terminator, the name of a field in the intended scrolled area must be specified, and if a string of values is specified, the values will be displayed on the top or bottom line of the scrolled area after the terminator is processed. If no argument is included, the call processes the last terminator that was used. The forms are: `CALL FPFT`*(term[,fid[,fval]])* `CALL FPFT`
FPUT	Displays the specified value in the specified field. The form is: `CALL FPUT`*(fval,fid[,fidx])*
FPUTAL	Displays values in all fields of the form. If a concatenated string of values is supplied, each value must be the same length as the field in which it is to be displayed and the values must be in the same order that the FGETAL call would produce for the form. Values from the supplied string are displayed in the first fields of the form, and defaults are displayed in any fields that remain. If no string of values is supplied, default values are displayed in all fields. The forms are: `CALL FPUTAL`*(fval)* `CALL FPUTAL`
FPUTL	If an argument is specified, displays the specified string on the bottom line of the screen. If no argument is specified, clears the bottom line. The forms are: `CALL FPUTL`*(fval)* `CALL FPUTL`
FRETAL	Returns the current values for all fields in the form in the same order that the FGETAL call would produce. The form is: `CALL FRETAL`*(fval)*
FRETN	Returns the current value of the specified field. The form is: `CALL FRETN`*(fval,fid[,fidx])*

Table 5-14. Listing of FORTRAN IV and FORTRAN-77 Form Driver Calls (Cont.)

Call Abbreviation	Summary and Forms
FSHOW	Clears the part of the screen that the specified form requires and displays the form with default field values. If a line number is specified, uses it as the starting line number for the form. The form is: `CALL FSHOW`*(fnam[,line])*
FSPOFF	Turns off the supervisor-only mode and allows the operator to enter and change data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The form is: `CALL FSPOFF`
FSPON	Turns on the supervisor-only mode and prevents the operator from entering or changing data in fields to which the Supervisor Only attribute was assigned with the Form Editor. The form is: `CALL FSPON`
FSTAT	Returns the status code for the last call that was processed as the value of the first argument. The value of the second argument is meaningful as an FCS or RMS system error code (depending on the version of the Form Driver in use) only if the value of the first argument is -4 or −18, indicating an error while trying to open or read a form library file. The form is: `CALL FSTAT`*(status[,stat2])*

5.11.3 Building a FORTRAN Task

Appendix D contains source listings of command files used to task-build FMS FORTRAN sample application programs.

5.12 The Interface for MACRO-11

In MACRO-11 programs, you can call the Form Driver with the instruction:

```
JSR PC,$FDV
```

$FDV is global

R0 must point to a list that includes both necessary and function-dependent arguments. Table 5-15 summarizes the Argument List.

Table 5-15. Offsets and Meanings of Necessary and Function-Dependent Arguments

Offset	Meaning
F$FNC	One-word function code.
F$REQ	One-word Required Argument List pointer.
F$NAM	One-word pointer to ASCIZ form library file specification or pointer to 6-byte ASCII form name, field name, or data name.
F$NUM	One-word starting line number to display form or index value for field or named data.
F$TRM	One-word field terminator code.
F$VAL	One-word data pointer.
F$LEN	One-word data length (in bytes).

F$ASIZ is the size in bytes of the Argument List.

NOTE

> The offsets to the arguments in the Argument List are defined as global symbols. The values for the offsets might change between releases of the software. For this reason, you should always refer to these offsets by name in your applications.

You must specify the $FDVDF macro in a .MCALL statement and invoke it in a MACRO task to define F$ASIZ to use the symbol to allocate space for the Argument List at assembly time. F$RSIZ and F$TSIZ are also defined by the $FDVDF macro. The $FDVDF macro is provided in the library.

You must specify a function code (F$FNC) and a pointer to the Required Arguments List (F$REQ) for each call to the Form Driver. You do not need to specify all the remaining arguments for each call. What arguments you must supply to the Form Driver depend on the call you are issuing (in other words, which function code you specify).

On return from the Form Driver, the carry bit is clear if the Form Driver completed the call successfully and the status code is positive (>0). All registers are preserved across a call to the Form Driver. If an error occurred, the carry bit is set and the Form Driver returns an error code in the status block. If you specified an invalid (undefined) function code, the error code FE$FCD is returned. Each call description in Chapter 6 includes the errors specific to that call.

The two necessary arguments in the Argument List are F$FNC and F$REQ. The other arguments listed in Table 5-15 are function-dependent arguments.

5.12.1 F$FNC, the MACRO-11 Function Code

Table 5-16 lists the MACRO-11 function codes for the different Form Driver calls and notes the corresponding high-level language calls.

In your application, you should always refer to the functions by the specified symbols in order to ensure compatibility with future versions of the software.

Table 5-16. MACRO-11 Function Codes and Meanings

Function Code	Meaning
FC$ALL	FGETAL - Get the responses for all fields.
FC$ANY	FGETAF - Get the response for any field that the user inputs.
FC$CIA	FCHIMP - Change active impure area.
FC$CLS	FLCLOS - Close form library.
FC$CSH	FCLRSH - Clear the entire screen and show the specified form.
FC$DAT	FIDATA & FNDATA - Get named data by index or by name.
FC$GET	FGET - Get the response for the specified field.
FC$GSC	FINLN - Get the current line of a scrolled area.
FC$LST	FPUTL - Output data to last line of display.
FC$OPN	FLOPEN - Open form library.
FC$PAL	FPUTAL - Output data to all fields.
FC$PSC	FOUTLN - Output data to the current line of a scrolled area.
FC$PUT	FPUT - Output data to a specified field.
FC$RAL	FRETAL - Return the contents of all fields.
FC$RTN	FRETN - Return the contents of the specified field.
FC$SHO	FSHOW - Show the specified form.
FC$SPF	FSPOFF - Turn supervisor-only mode off.
FC$SPN	FSPON - Turn supervisor-only mode on.
FC$TRM	FPFT - Process field terminator.

5.12.2 F$REQ, Required Argument List Pointer

Table 5-17 lists the contents of the Required Argument List to which F$REQ must point.

Table 5-17. Required Argument List Offsets and Meanings

Offset	Meaning
F$STS	One-word status block pointer.
F$CHN	One-word channel number (LUN) for form I/O.
F$IMP	One-word pointer to the impure area provided for the Form Driver.

F$RSIZ is the size in bytes of the Required Argument List.

You must use the $FDVDF macro to define F$RSIZ at assembly time. F$RSIZ should be used to allocate space for the Required Argument List.

5.12.2.1 F$STS, the Status Block Pointer — This word points to a two-word status block that the Form Driver maintains for each active call. The first word of the Status Block reflects the status of a call to the Form Driver as follows:

>0 - Successful completion
<0 - Error encountered

Table 5-1 in Section 5.1.1 lists the MACRO-11 status codes and their meanings.

Your MACRO-11 application should always use the global symbols listed in Table 5-1 to ensure compatibility with future versions of the software. Values are also given for each symbol because it is not possible to refer to the global symbols from high-level language programs.

If the first word of the status block contains FE$IOL or FE$IOR, the code corresponding to the specific error is returned in the second word. See RMS documentation for a list of these error codes. (MACRO-11 tasks can use either RMS or FCS, depending on which one you have configured in your version of the Form Driver.)

5.12.2.2 F$CHAN, the Form Channel Number — This word contains the channel number (LUN) information that the Form Driver uses to access a form library.

5.12.2.3 F$IMP, the Impure Area Pointer

— This word points to the beginning of the impure area available to the Form Driver. The impure area stores:

- Information pertaining to the current form.
- Operator responses in the context of that form.

The first word of the impure area must contain its total length in bytes. Both the Form-Wide Attributes Questionnaire of the Form Editor and the listing of your form produced by the Form Utility (FUT) in response to the /FD option tell you the length of the necessary impure area.

To preserve the context of a form, the impure area pointer must be the same for all calls to the Form Driver that refer to that form. Your program cannot modify the impure area, which is reserved for use by the Form Driver.

5.12.3 Function-Dependent Arguments

Each function code can require a different set of function-dependent arguments. Chapter 6 lists the required and optional arguments in the file descriptions of each call.

Table 5-18 summarizes all MACRO-11 function-dependent arguments. The following sections describe these arguments in detail.

5.12.3.1 F$NAM, the Name Pointer

— This argument points to various names, depending on which call is associated with them.

In a call to open a form library, F$NAM on input contains a pointer to the file specification for the form library. The file specification must be an ASCII string terminated with a null.

In a call to display a form, F$NAM on input contains a pointer to a 6-byte ASCII form name. The name must be left-justified, with unused positions blank filled.

In a call that requires a field name, F$NAM points to the 6-byte ASCII field name associated with the field when the form was defined.

In a call to get named data, F$NAM points to a 6-byte ASCII data name that you specified using the Form Editor.

5.12.3.2 F$NUM, the Line Number and Field Index — F$NUM can contain two kinds of information, depending on whether the call displays a form or requires a field name.

In a call to display a form, F$NUM on input contains a line number from 0 to 23, specifying display of the first line to be cleared.

In calls that require a field name, F$NUM is used to pass a field index identifying the specific field between the Form Driver and the application task. The field index is a positive integer specifying the element referred to for a field defined as an array. For fields not defined as arrays, the Form Driver ignores the ındex value and returns it to the application task as 1.

5.12.3.3 F$VAL, the Data Value Pointer — In a call to output data (to a specified field, to all fields, or to the last line) F$VAL on input contains a pointer to the data to be displayed. If the corresponding length of data (F$LEN) is zero, the Form Driver ignores F$VAL and uses default values.

On returning from a call to get data (get a field, get any field, get all fields, return a specified field or all fields, or get named data), $FVAL contains a pointer into the Form Driver impure area to the data requested.

5.12.3.4 F$LEN, the Data Length — You must provide the data length in bytes for all calls to output data. The Form Driver uses a length greater than zero as the length of the data to output. The F$VAL argument points to this data.

The Form Driver uses a length less than zero as an indicator that the data ends with a null. See descriptions of individual calls for use of a length of zero, which specifies that default values are to be used.

For all calls to get data, the Form Driver returns the data length in bytes in F$LEN. Responses returned to the calling task are always the length of the field (minus field-marker characters) and are blank-filled or zero-filled according to the field definition.

5.12.3.5 F$TRM, the Field Terminator Code — F$TRM is one word in which the Form Driver returns to your task an integer code for the key that the operator used to terminate a field entry. The argument is also input for the FPFT call.

In Section 5.2.1, Table 5-2 summarizes the field terminator keys, codes, and meanings. In Section 5.2.2, Table 5-4 summarizes the codes for the alternate keypad mode terminators. Values of the terminators are given for use by high-level language tasks, which cannot refer to global symbols. These are also the terminators that the FPFT call processes.

5.12.4 Keyword Encoded Macros

To simplify the interface to the Form Driver for the MACRO-11 programmer, the software provides the following keyword macro in the macro library FMSMAC.MLB:

```
$FDV ARG,FNC,REQ,NAM,NUM,TRM,VAL,LEN
```

Table 5-18 summarizes the requirements and meanings of the keywords and shows the relationships between the keywords on input and the global offsets for function-dependent arguments on output. The high-level language arguments are also listed in Table 5-18 to show the correspondence between them and their MACRO-11 counterparts.

Table 5-18. Summary of Arguments, Keywords, and Offsets for High-Level Language and MACRO-11 Form Driver Calls

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Argument List.
CHAN	* None	A channel number for a form library file.
FID	NAM	A field name or a named data label, six characters long, including padding (for FORTRAN IV and FORTRAN-77, add a null byte also.) For MACRO-11 only, a pointer to a 6-byte ASCII field name or named data label. To specify a scrolled area, use the name of any field in the scrolled area.
FIDX	NUM	A field index for the specified field (when the field is indexed) or the index for a named data value.
	LEN	The total length of the data to be displayed. The value must be −1 for ASCIZ strings and 0 for restoring default values or clearing the last line of the screen. (An input value for MACRO-11 only.)
FLNM	NAM	A form library file specification. For MACRO-11 only, a pointer to an ASCIZ form library file specification.
FNAME	NAM	A form name, 6 characters long, including padding (for FORTRAN IV and FORTRAN-77, add a null byte also). For MACRO-11 only, a pointer to a 6-byte ASCII form name.

Table 5-18. Summary of Arguments, Keywords, and Offsets for High-Level Language and MACRO-11 Form Driver Calls (Cont.)

High-Level	MACRO-11	Requirement or Value
FVAL	VAL	As an input value, the single value or the concatenated values to be displayed: • in a field. • in the top, bottom, or current line of a scrolled area. • in the last line of the screen. • in an entire form.
IMPURE	* None	The name of a subscripted variable (or array) of bytes for the impure area.
LINE	NUM	The explicit starting line for the form, overriding the line number assigned with the Form Editor.
TERM	TRM	As an input value, the numeric code for the terminator that the Form Driver is to process.

Outputs

The Status Code is set for all calls. Most calls are processed by the Form Driver, and for these (with MACRO-11 only) R0 points to the Argument List when the call processing is complete.

FID	F$NAM(R0)	A named data label or the current field name. For MACRO-11 only, a pointer to a named data label or data name.
FIDX	F$NUM(R0)	A field index.
FLEN	F$LEN(R0)	For the FLEN high-level language call, the length of a specified field (not the length of the data the field contains). For MACRO-11 calls, the total length of the fields for which values are returned by the Form Driver, such as: • a named data value. • a single field. • all fields in one line of a scrolled area. • all fields in a form.
FVAL	F$VAL(R0)	A named data value, a single field value, or a concatenated string that is composed of several field values (including padding when a value is shorter than its field). For MACRO-11 only, a pointer to the value or string in the impure area.

Table 5-18. **Summary of Arguments, Keywords, and Offsets for High-Level Language and MACRO-11 Form Driver Calls (Cont.)**

High-Level	MACRO-11	Requirement or Value
TERM	F$TRM(R0)	The numeric code for the key that the operator used to terminate input: • in a field. • in a line in a scrolled area. • in an entire form.
STATUS	* None	A numeric code for the completion status of the last call that was executed.
STAT2	* None	A numeric FCS or RMS status code for detailed information when the STATUS value is –4 or –18 (for MACRO-11, the equivalent codes are FE$IOL and FE$IOR.)

* For MACRO-11 only, the channel number and pointers to the impure area and status block must be specified in the Required Argument List.

The $FDV macro should be used in accordance with the keyword calling convention of MACRO-11. The argument names correspond to the last three characters of the Global Argument List offsets defined by the Form Driver.

The following information should enable you to use the $FDV macro in most instances. For further details about calling macros with keyword arguments, see the *PDP-11 MACRO-11 Language Reference Manual.*

All arguments to the $FDV macro must be in the form of instruction source operands to be used in MOV instructions on the source operand side as shown below:

MOV *arg,destination*

The destination of all the arguments is either R0, in the case of ARG, or F$xxx(R0) in the case of all other arguments, where xxx is the name of the argument. Any arguments that do not appear in the macro call are not changed in the Argument List by the macro expansion.

Since the list of arguments to the macro can be quite long, a single call to the Form Driver can be broken up into several calls to the macro. If the FNC argument specifying the function code is missing, the call to the Form Driver is not generated, and the next call to the macro can fill in further arguments. Argument blocks should be built at execution time or by symbolic means at assembly time.

The following macro call loads the Required Argument List pointer into the Argument List. Loading the pointer normally needs to be done only once in a program.

$FDV ARG=*arglst*,REQ=*reqlst*

As a result of this call, R0 points to the specified Argument List. The Form Driver is not called by the macro call.

The following variations on a call to the Form Driver to write data to a specified field (FPUT) illustrate the syntax of the $FDV macro:

Example 1

```
MOV     #REQLST,R0                  ; REQUIRED ARGUMENTS LIST
                                    ; POINTER

MOV     #STAT,F$STS(R0)             ; STATUS BLOCK POINTER
MOV     #1,F$CHN(R0)                ; LIBRARY CHANNEL NUMBER
MOV     #IMPURE,F$IMP(R0)           ; IMPURE AREA POINTER
$FDV    ARG=#ARGLST,REQ=#REQLST     ; PUT REQ ARG
                                    ; LIST PTR IN ARG LIST
$FDV    ARG=#ARGLST,FNC=PUT,
VAL=#BUFFER,LEN=DATLN,
NAM=#FLD1,NUM=#1
```

In this call, the Argument List is ARGLST, the function is FC$PUT, the data to output is in BUFFER, the data length is in DATLN, the field name is in FLD1, and the index is 1. The following data are associated with the call.

```
          .
          .
          .
ISIZ = 1024.
BUFFER: .ASCII  /TEN CHARS?/  ;DATA
DATLN:  .WORD   10.           ; LENGTH OF DATA
FLD1:   .ASCII  /FIELD1/      ; FIELD NAME (6 CHARS)
ARGLST: .BLKB   F$ASIZ
                              ; ALLOCATE SPACE FOR
                              ; ARGUMENT LIST
REQLST: .BLKB   F$RSIZ
                              ; ALLOCATE SPACE FOR
                              ; REQ ARG LST
IMPURE: .WORD   ISIZ
                              ; SIZE OF IMPURE AREA IN BYTES
.BLKB   ISIZ-2                ; THE IMPURE AREA
STAT:   .BLKW   2             ;2 WORD STATUS BLOCK
          .
          .
          .
```

Example 2

The call in Example 1 can be broken into two lines as follows:

```
$FDV   ARG=#ARGLST,VAL=#BUFFER,LEN=DATLN
$FDV   FNC=PUT,NAM=#FLD1,NUM=#1
```

After the first call to the macro, the Form Driver is not called because the function code (FNC=PUT) is not included. It is the second call to the Form Driver that, in addition to providing values for further arguments, calls the Form Driver.

Example 3

In the following example of an FPUT call, R0 is assumed to point to the argument block.

1. The field name is in FLD1.
2. The index is in R2.
3. R1 points to the data to be output.
4. R3 points to the length.
5. R3 is advanced by the call.

```
$FDV   FNC=PUT,NAM=#FLD1,NUM=R2,VAL=R1,LEN=(R3)+
```

This format for issuing the macro call is an alternative to those shown in the first two examples.

Chapter 6 provides a detailed description of each call to the Form Driver.

5.12.5 Special Information for I/O from a MACRO-11 Program

A MACRO-11 program can use either RMS or FCS. The program must initialize FCS before calling the Form Driver if media-resident forms are used. (See the *RSX-11M I/O Operations Manual*, Chapter 2 for details.) If RMS is chosen, the program must initialize RMS with a call to the $INIT or $INITIF macros if media-resident forms are used.

To avoid loss of typeahead in MACRO-11 programs using the Form Driver, attach the terminal with the QIO system directive.

Example: `QIOW$S #IO.ATT,#5,#5`

5.12.6 Program Sections Used by FMS

The following Program Section (PSECT) names are reserved for use by FMS.

PSECT	Usage
$$FMS	Task specific data and buffers
$$FMSV	Offsets to vector area
$$FMSB	Offsets to buffer descriptor
$$FMS1	Data areas for form driver
$FIDX$	Memory-resident form index
$FIDY$	End of memory-resident form index
$FORM$	Memory-resident form descriptions
.FDV.	Form driver and support code
$HLDAT	High-level language data area
$HLEXE	High-level language call definitions
.ERR.	Error messages
.DBG.	Debug error messages

Three PSECTs are used to keep information about memory-resident forms.

$FIDX$	Form Index
$FIDY$	End of Form Index
$FORM$	Form Descriptions

In the Form Driver data area (FDVDAT) there is a pointer to the form index PSECT $FIDX$. FDVDAT must be the first contribution to this PSECT. That is, it must be referenced in the task-build command before any of the memory-resident forms. PSECTs must be ordered such that $FIDX$ and $FIDY$ are adjacent. This is normally the case since PSECTs are usually ordered alphabetically.

Any of these PSECTs can be in overlays, but $FIDX$ and $FIDY$ are normally in the root. It is quite reasonable to place forms in overlays. The index entries will be pulled into the root, since $FIDX$ and $FIDY$ have the GBL attribute.

5.12.7 Form Driver Conditionals

All Form Driver conditionals are in the file FSYCND.MAC. The values of the following conditionals are set by the configuration procedure.

Symbol	Default	Meaning
BUFS$B	1	Buffer size in blocks.
DRB$	1	Number of directory buffers.
FDB$N	1	Number of form libraries that can be open simultaneously.
RMSI$0	0	0 for FCS 1 for RMS

Terminal support code varies for the systems that FMS supports. This conditional selects the terminal service required.

Symbol	Value	Meaning
TIOT$P	0	RSX-11M Half-duplex driver (V3.1)
	1	RSX-11M, M+ Full-duplex driver.
	2	Reserved.
	4	Reserved.

The default is 1.

The Form Driver uses several conditionals for internal processing of event flags and I/O management. The following table lists the conditionals and their default values.

Symbol	Default	Meaning
RESL$B	0	FDVDAT conditional data and offsets.
	1	Library offsets.
	2	Data only for task.
IOE$F	32	Event flag for form library I/O.
TLU$N	5	Terminal LUN.
TEF$N	31	Terminal output event flag.
TIEF$N	30	Terminal input event flag.

5.12.8 Event Flags

The terminal event flags (TEF$N and TIEF$N) are selected to not interfere with event flags used by the task for other purposes.

5.12.9 Building a MACRO-11 Program

When assembling MACRO-11 programs, include the macro library.

```
FMSMAC/ML
```

In the command line to resolve MACRO-11 definitions for $FDV and other FMS macros.

Appendix D contains a sample task-building command file that illustrates task-building a MACRO-11 application program.

Chapter 6 Form Driver Calls

A flexible set of Form Driver calls provides functions that display forms, display data in fields, and handle terminal input. (Additional calls are provided for high-level language programs only.) The descriptions describe what each call does, the input arguments it requires, and the output it returns to your task. For each call, syntax in high-level languages and MACRO-11 is indicated.

See Chapter 5 for information on each programming language.

6.1 FCHIMP – Change the Impure Area

The FCHIMP call changes the impure area pointer within the FMS vector area. The impure area must be initialized with the FINIT call before you can use the FCHIMP call.

When you use FCHIMP, the library channel number is moved from the old impure area to the new impure area.

BASIC-PLUS-2 and FORTRAN

`CALL FCHIMP`*(impure)*

COBOL

`CALL "FCHIMP" USING BY DESCRIPTOR` *impure.*

DIBOL-83

`XCALL FCHIMP`*(impure)*

MACRO-11

`$FDV ARG=`*arglst,*`FNC=CIA,REQ=`*reqlst,*`VAL=`*impure*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
IMPURE	* None	The name of a subscripted variable (or array) of bytes for the impure area to change to.

Outputs

Positive return status values imply a new impure area is now in use. Negative return status values imply the old impure area is still in use.

*For MACRO-11 only, the pointers to the impure area and status block are contained in the Required Arguments List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 2	FE$IMP	Impure area too small.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)

6.2 FCLRSH – Clear Entire Screen and Display Form

The Form Driver clears the entire screen and displays the specified form as described under FSHOW.

BASIC-PLUS-2 and FORTRAN

CALL FCLRSH(*fnam[,line]*)

COBOL

CALL "FCLRSH" USING BY DESCRIPTOR *fnam*[,BY REFERENCE *line*].

DIBOL-83

XCALL CLRSH(*fnam[,line]*)

MACRO-11

$FDV ARG=*arglst*,FNC=CSH,REQ= *reqlst*,NAM=*form*,NUM=*line*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Argument List.
FNAM	NAM	A form name. For MACRO-11 only, a pointer to a 6-byte form name.
LINE	NUM	The explicit first line for the form, overriding the line number assigned with the Form Editor.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
− 6	FE$ICH	Invalid channel number specified.
− 7	FE$FCH	Form library not open on specified channel.
− 8	FE$FRM	Invalid form definition.
− 9	FE$FNM	Specified form does not exist.
−10	FE$LIN	Invalid first line number to display form.
−18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
−20	None	Wrong number of arguments in call. (For high-level language programs only.)
−21	None	Impure area not yet initialized. (For high-level language programs only.)

6.3 FGCF - Return the Current Field Name

The FGCF call does not call the Form Driver. It merely returns arguments from the Form Driver argument block. FGCF is to be used in conjunction with the FGET call to allow immediate processing of fields and immediate feedback to the terminal operator (as well as implementing scrolling).

In a high-level language task, this call must follow a call to process a field terminator if you wish to access the name of the new current field and its index value.

There is no MACRO-11 equivalent for this call.

BASIC-PLUS-2 and FORTRAN

`CALL FGCF` *(fid)*

COBOL

`CALL "FGCF" USING BY DESCRIPTOR` *fid*`[,BY REFERENCE` *fidx*`].`

DIBOL-83

`XCALL FGCF`*(fid[,fidx])*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
None	–	
Outputs		
FID	–	The field name for the current field. (An output value for high-level languages only.)
FIDX	–	The field index for the current field (when that field is an indexed field). (An output value for high-level languages only.)

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.4 FGET – Get the Value From the Specified Field

The Form Driver places the cursor in the initial position of the specified field and accepts input by the operator in that field. (Section 4.2.2.2 describes the initial cursor position for fields that have different attributes.) When the Form Driver returns control to the application, the specified field is the current field.

If the first character of the field name specified is an asterisk (*), the Form Driver sets as the current field the first field in the form that is not display-only and not within a scrolled area. The field name and index value for that field are returned to your task. A high-level language program must use the FGCF call to get the field name and index.

If you do not specify a field name, the Form Driver places the cursor in the lower right corner of the screen and waits for the operator to press the ENTER key, indicating readiness to proceed; at that point, the field terminator code 0 (or, for MACRO-11 only, FT$NTR) is returned to your program. This "special get" call serves to synchronize the operation of your program with the pace of the terminal operator.

BASIC-PLUS-2 and FORTRAN

CALL FGET*(fval,term,fid[,fidx])*

CALL FGET

COBOL

```
CALL "FGET" USING BY DESCRIPTOR fval,BY REFERENCE term,
                  BY DESCRIPTOR fid[,BY REFERENCE fidx].
```

DIBOL-83

XCALL FGET*(fval,term,fid[,fidx])*

XCALL FGET

CALL "FGET"

MACRO-11

$FDV ARG=*arglst*,FNC=GET,REQ=*reqlst*,NAM=*fld*,NUM=*idx*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
FID	NAM	A field name. For MACRO-11 only, a pointer to a 6-byte ASCII field name.
FIDX	NUM	A field index for the specified field (when the field is indexed).
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
–	F$LEN(R0)	The length of the field. (An output value for MACRO-11 only.)
–	F$NAM(R0)	When the input field name begins with an asterisk (*), a pointer to the 6-byte ASCII name of the field name for the first field that is not display-only and not within a scrolled area. (An output value for MACRO-11 only.)
–	F$NUM(R0)	When the input field name begins with an asterisk (*), the field index for the first field that is not display-only and not within a scrolled area. (An output value for MACRO-11 only.)
FVAL	F$VAL(R0)	The field value, including padding.
TERM	F$TRM(R0)	The numeric code for the key the operator used to terminate input in the field.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 6	FE$ICH	Invalid channel number specified.
– 7	FE$FCH	Form library not open on specified channel.
–11	FE$FLD	Specified field does not exist (invalid field name or index).
–12	FE$NOF	No fields defined for current form.
–13	FE$DSP	GET call illegal for display-only field(s).
–18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.5 FGETAF – Get the Value for Any Field

The Form Driver waits for the operator to respond to any field. The operator can move the cursor to any field that is not display-only. The Form Driver accepts as valid responses either the ENTER key alone or any field terminator entered in a field the operator has modified. The field that is entered becomes the current field.

The FGETAF call is invalid for a form that contains a scrolled area.

BASIC-PLUS-2 and FORTRAN

```
CALL FGETAF(fval,term,fid[,fidx])
```

COBOL

```
CALL "FGETAF" USING BY DESCRIPTOR fval,BY REFERENCE term,
                    BY DESCRIPTOR fid[,BY REFERENCE fidx].
```

DIBOL-83

```
XCALL GETAF(fval,term,fid[,fidx])
```

MACRO-11

```
$FDV ARG=arglst,FNC=ANY,REQ=reqlst
```

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Argument List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
FID	F$NAM(R0)	The field name for the field in which the operator responds. (For MACRO-11 only, a pointer to the 6-byte ASCII field name.)
FIDX	F$NUM(R0)	The field index for the field in which the operator responds (when that field is an indexed field).
	F$LEN(R0)	The length of the field.
FVAL	F$VAL(R0)	The field value, including padding.
TERM	F$TRM(R0)	The numeric code for the key that the operator used to terminate input in the field.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 6	FE$ICH	Invalid channel number specified.
– 7	FE$FCH	Form library not open on specified channel.
–12	FE$NOF	No fields defined for current form.
–13	FE$DSP	GET call illegal for display-only field(s).
–18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.6 FGETAL - Get All Field Values

Beginning with the first field in the form, the terminal operator can move around in the form and enter and change values in any fields he or she chooses. The Form Driver waits for the operator to press the ENTER key as a signal of completion of the entire form. The values of all fields are then returned to your program as a concatenated string of the default values and new entries that are displayed. Fields are returned in left-to-right, top-to-bottom order, except when a form contains vertically indexed fields. (See Section 4.1.7 for a description of the order of return for indexed fields.)

Normally, when a form includes fields with the Response Required or Must Fill attribute, all fields of the form must be completed before the Form Driver will return to the program. Otherwise, a message is displayed, the bell rings, and the cursor is located at the first incomplete field.

However, when the program has set the terminal to the alternate keypad mode, the alternate keypad mode terminators are always passed immediately to the program. (See Section 4.2.2 for a description of the alternate keypad mode feature.)

The FGETAL call with no arguments only stores the values for all fields in the Form Driver impure area. You can then access the values with the FRETN or FRETAL call. The calls are described later in this chapter.

The FGETAL call is invalid for a form that contains a scrolled area.

BASIC-PLUS-2 and FORTRAN

CALL FGETAL*(fval[,term])*

CALL FGETAL

COBOL

CALL "FGETAL" USING BY DESCRIPTOR *fval*[,BY REFERENCE *term*] .

DIBOL-83

XCALL GETAL*(fval[,term])*

XCALL GETAL

CALL "FGETAL"

MACRO-11

$FDV ARG=*arglst*,FNC=ALL,REQ=*reqlst*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Argument List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
–	F$LEN(R0)	The total length for all fields in the form. (An output value for MACRO-11 only.)
FVAL	F$VAL(R0)	The concatenated values for all fields in the form. For MACRO-11 only, a pointer to the concatenated values in the impure area.
TERM	F$TRM(R0)	The numeric code for the key the operator used to terminate input in the form.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 6	FE$ICH	Invalid channel number specified.
– 7	FE$FCH	Form library not open on specified channel.
–11	FE$FLD	Specified field does not exist (invalid field name or index).
–12	FE$NOF	No fields defined for current form.
–13	FE$DSP	GET call illegal for display-only field(s).
–18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can found with the FSTAT call and is returned in the second word of the status block).
–19	FE$IFN	Specified call invalid in current context of form.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.7 FIDATA – Get Named Data by Index

This call accesses named data by using the index into the named data rather than the name of the data.

BASIC-PLUS-2 and FORTRAN

CALL FIDATA(*fidx,fval[,fid]*)

COBOL

```
CALL "FIDATA" USING BY REFERENCE fidx,BY DESCRIPTOR fval
                   [,BY DESCRIPTOR fid].
```

DIBOL-83

XCALL IDATA(*fidx,fval[,fid]*)

MACRO-11

$FDV ARG=*arglst*,FNC=DAT,REQ=*reqlst*, NAM=#0,NUM=*num*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Argument List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
FID	F$NAM(R0)	A pointer to the named data label in the impure area. (An output value for MACRO-11 only.)
-	F$LEN(R0)	The length of the named data value. (An output value for MACRO-11 only.)
FVAL	F$VAL(R0)	The named data value for the index requested. For MACRO-11 only, a pointer to the named data value in the impure area.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–15	FE$DNM	Named data specified does not exist.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.8 FINIT – Initialize Impure Area for High-Level Language Tasks

The FINIT call initializes the space for the impure area. The FINIT call must precede any other Form Driver calls in a program.

For the size of the impure area, add 64 bytes to the size reported by the Form Editor for the largest form that your program uses. The high-level language interface uses the first 64 bytes of the impure area for the Argument List for Form Driver calls.

BASIC-PLUS-2 and FORTRAN

`CALL FINIT`*(impure, size,trmctl[,status])*

COBOL

```
CALL "FINIT" USING BY DESCRIPTOR impure, BY REFERENCE size,
                   BY DESCRIPTOR trmctl[ ,BY REFERENCE status].
```

DIBOL-83

`XCALL FINIT`*(impure,size,trmctl[,status])*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
IMPURE	* None	The name of a subscripted variable (or array) of bytes for the impure area.
SIZE		The size of the impure area.
Outputs		
STATUS	* None	A numeric code for the completion status of the call.

* For MACRO-11 only, the pointers to the impure area and status block are contained in the Required Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 2	FE$IMP	Impure area too small.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)

6.9 FINLN – Get Current Line of Scrolled Area

Within the current line of the specified scrolled area, the Form Driver usually places the cursor at the initial position of the first field that is not display-only. However, if the last call to the Form Driver was an FPFT call to process the terminator to scroll backward to the previous field (value = 7, MACRO-11 global = FT$SPR), the cursor is placed at the initial position of the last field on the line that is not display-only. The terminal operator can complete the line to his or her satisfaction. The Form Driver then returns the contents of the line as a concatenated string of field values.

BASIC-PLUS-2 and FORTRAN

CALL FINLN*(fid, fval[,term])*

COBOL

```
CALL "FINLN" USING BY DESCRIPTOR fid,
                   BY DESCRIPTOR fval[ ,BY REFERENCE term] .
```

DIBOL-83

XCALL FINLN*(fid,fval[term])*

MACRO-11

$FDV ARG=*arglst*,FNC=GSC,REQ= *reqlst*,NAM=*fld*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Argument List.
FID	NAM	A field name for any field within the scrolled area to be processed. For MACRO-11 only, a pointer to a 6-byte ASCII field name.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
–	F$LEN(R0)	The total length of all fields in the line. (An output value for MACRO-11 only.)
FVAL	F$VAL(R0)	The values for all fields in the line, concatenated from left to right. For MACRO-11 only, a pointer to the concatenated values in the impure area.
TERM	F$TRM(R0)	The numeric code for the key the operator used to terminate input in the line.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
− 6	FE$ICH	Invalid channel number specified.
− 7	FE$FCH	Form library not open on specified channel.
−11	FE$FLD	Specified field does not exist (invalid field name or index).
−14	FE$NSC	Specified field not in scrolled area.
−18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
−20	None	Wrong number of arguments in call. (For high-level language programs only.)
−21	None	Impure area not yet initialized. (For high-level language programs only.)
−22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.10 FLCHAN – Set I/O Channel (LUN) for Form Library File

The FLCHAN call sets the I/O channel (LUN) to open or access a form library file. The FLCHAN call must be issued after the FINIT call and before the first FLOPEN call.

The FLCHAN call has two uses: it sets the channel for the next FLOPEN call, and it sets the channel on which forms will be accessed by FSHOW and FCLRSH calls. More than one form library file can be open at one time by using FLCHAN to switch channels between FLOPEN calls. Forms can be selected from one of several libraries by using FLCHAN to select the desired library before the FSHOW or FCLRSH call. The library channel should not be switched until just before the next call to show a form, since operator HELP or SCREEN REFRESH functions require reaccessing of the form description for the current form the operator is using.

The channel named must be legal for the program and must have been designated as such when the program was built.

BASIC-PLUS-2 and FORTRAN

`CALL FLCHAN` *(chan)*

COBOL

`CALL "FLCHAN" USING BY REFERENCE` *chan.*

DIBOL-83

`XCALL LCHAN`*(chan)*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
CHAN	* None	A channel number for a form library file.
Outputs		
None	–	

* For MACRO-11 only, the channel number is specified in the Required Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.11 FLCLOS - Close Form Library

The Form Driver closes the form library open on the current channel (LUN).

BASIC-PLUS-2 and FORTRAN

```
CALL FLCLOS
```

COBOL

```
CALL "FLCLOS".
```

DIBOL-83

```
XCALL LCLOS
```

MACRO-11

$FDV ARG=*arglst*,FNC=CLS,REQ=*reqlst*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Argument List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 6	FE$ICH	Invalid channel number specified.
– 7	FE$FCH	Form library not open on specified channel.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.12 FLEN – Return the Length of the Specified Field

The FLEN call returns the length of the specified field. The high-level language forms of the call are the only ones that exist. No MACRO-11 equivalent is supplied.

BASIC-PLUS-2 and FORTRAN

CALL FLEN*(flen,fid[,fidx])*

COBOL

```
CALL "FLEN" USING BY REFERENCE flen,BY DESCRIPTOR fid
                  [,BY REFERENCE fidx].
```

DIBOL-83

XCALL FLEN*(flen,fid[,fidx])*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
FID	-	A field name.
FIDX	-	A field index for the specified field (when the field is indexed).
Outputs		
FLEN	-	The field length.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.13 FLOPEN – Open Form Library

The Form Driver opens the specified form library file on the current form library channel. The channel (LUN) must be valid for the task and not attached by the program for another use. Except for COBOL, which uses LUN 1, LUN 5 is the default used for terminal service by the Form Driver and is therefore not available to your program. DIBOL-83 requires a trailing space for the argument "flnm". The form library file specification must have the following format (optional elements are enclosed in square brackets, but the UIC, if specified, must be enclosed in square brackets).

```
[device:][UIC]file.typ[;version]
```

The default file type for a form library file is .FLB.

BASIC-PLUS-2 and FORTRAN

CALL FLOPEN*(flnm)*

COBOL

CALL "FLOPEN" USING BY DESCRIPTOR *flnm.*

DIBOL-83

XCALL LOPEN*(flnm)*

MACRO-11

$FDV ARG=*arglst,*FNC=OPN,REQ=*reqlst,*NAM=*lib*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
FLNM	NAM	A form library file specification. For MACRO-11 only, a pointer to an ASCIZ form library file specification.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 3	FE$FSP	Invalid file specification.
– 4	FE$IOL	Error encountered opening form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
– 5	FE$FLB	Specified file not form library.
– 6	FE$ICH	Invalid channel number specified.
–18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.14 FNDATA – Get Named Data by Name

This call is used to access, by name, data that has previously been associated with a form as named data. The Form Driver can access the named data that is attached to a form description but does not display the data with the form.

You can use the FSTAT call to determine whether the FNDATA call returns a valid named data value. If the FSTAT call returns a status value of –15, no named data value was found.

BASIC-PLUS-2 and FORTRAN

CALL FNDATA*(fid,fval)*

COBOL

CALL "FNDATA" USING BY DESCRIPTOR *fid,* BY DESCRIPTOR *fval.*

DIBOL-83

XCALL NDATA*(fid,fval)*

MACRO-11

$FDV ARG=*arglst,*FNC=DAT,REQ=*reqlst,*NAM=*nam*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
FID	NAM	A named data label. For MACRO-11 only, a pointer to a 6-byte ASCII named data label.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
-	F$LEN(R0)	The length of the named data value. (An output value for MACRO-11 only.)
FVAL	F$VAL(R0)	The named data value for the label requested. For MACRO-11 only, the pointer to the named data value in the impure area.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–15	FE$DNM	Specified named data does not exist.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.15 FOUTLN – Output Data to Current Line of Scrolled Area

The Form Driver outputs the specified data to the current line of the scrolled area. The scrolled area is identified by specifying the name of any field in that area.

If the data is too long for the line, the Form Driver returns an error to your program and truncates the data when it is displayed. If the data is too short for the line, default values are displayed for fields for which no data is provided. If the length of the data is zero, the Form Driver restores default values to all fields in the current line of the scrolled area.

The Form Driver does not validate data output to fields from the application task. This is true for both explicit output and default values.

BASIC-PLUS-2 and FORTRAN

CALL FOUTLN*(fid[,fval])*

COBOL

CALL "FOUTLN" USING BY DESCRIPTOR *fid*[,BY DESCRIPTOR *fval*].

DIBOL-83

XCALL OUTLN*(fid[,fval])*

MACRO-11

$FDV ARG=*arglst*,FNC=PSC,REQ= *reqlst*,NAM=*fld*,VAL=*val*, LEN=*len*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
FID	NAM	A field name for any field within the scrolled area to be processed. For MACRO-11 only, a pointer to a 6-byte ASCII field name.
-	LEN	The total length of the data to be displayed (must be –1 for ASCIZ strings and 0 for restoring the default values to all fields in the line). (An input value for MACRO-11 only.)
FVAL	VAL	The field value(s) to be displayed in the current line of the scrolled area.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–11	FE$FLD	Specified field does not exist (invalid field name or index).
–14	FE$NSC	Specified field not in scrolled area.
–16	FE$DLN	Data specified for output too long (truncated by Form Driver).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.16 FPFT - Process the Field Terminator

The Form Driver processes the specified field terminator. Then, if the field terminator is valid, the Form Driver returns the name and index for the new current field to your program. In high-level languages, the FGCF call must be issued immediately after the FPFT call to get the name of the new current field.

If the specified field terminator is the ENTER key, the Form Driver checks the form for Response Required and Must Fill fields. If the form contains such fields and the requirements are not met for every field in the form, the status code (as shown below) is set to show that the form is incomplete.

In MACRO-11 tasks, the Form Driver returns the pointer to the name of the first incomplete field and the index value of that field. In high-level language programs, the FGCF ("Get the Name of the Current Field") call is used to obtain the field name and index value.

If the field terminator indicates scrolling, you must provide the name of a field in the call. The field name identifies the scrolled area the Form Driver is to manipulate. The name of any field in the scrolled area is sufficient.

If the field terminator indicates the TAB or DOWNARROW key, your task can also specify data to be displayed in the bottom line of the scrolled area when the area is scrolled forward.

If no data is specified and the current line is not the bottom line of the scrolled area, the cursor moves down one line and that line becomes the new current line. If no data is specified and the current line is the bottom line, the area is scrolled up and default values restored to the bottom line. If data is specified, the area is always scrolled up, the data displayed on the bottom line, and the current line remains the same line. If the terminator is for the BACKSPACE or UPARROW key, your task can specify the data to be displayed in the top line of the scrolled area when the area is scrolled backward. If no data is specified and the current line is not the top line of the scrolled area, the cursor moves up one line and that line becomes the new current line. If no data is specified and the current line is the top line, the area is scrolled down and default values are restored to the top line. If data is specified, the area is always scrolled down, the data displayed on the top line, and the current line remains the same.

If a field terminator is not specified in the call, the last field terminator returned from the Form Driver is processed.

BASIC-PLUS-2 and FORTRAN

```
CALL FPFT(term[,fid[,fval]])
```

or

```
CALL FPFT
```

COBOL

```
CALL "FPFT" USING BY REFERENCE term
                  [,BY DESCRIPTOR fid]
                  [,BY DESCRIPTOR fval].
```

DIBOL-83

```
XCALL FPFT (term[,fid[,fval]])
```

MACRO-11

```
$FDV   ARG=arglst,FNC=TRM,REQ=reqlst,   TRM=trm,NAM=fld,
             VAL=val,LEN=len
```

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
FID	NAM	A field name, identifying the scrolled area to be processed. For MACRO-11 only, a pointer to a 6-byte ASCII field name. (An input value only if a scrolled area terminator is specified.)
–	LEN	The length of the data to be displayed (must be −1 for ASCIZ strings and 0 if no data is specified). (An input value for MACRO-11 only and only if a scrolled area terminator is specified.)
FVAL	VAL	The field value(s) to be displayed in the top or bottom line of the scrolled area. (An input value only if a scrolled area terminator is specified.)
TERM	TRM	A numeric code for the terminator that the Form Driver is to process.
Outputs		
None	None	The status code is set. (For MACRO-11 only, R0 points to the Argument List.)
–	F$NAM(R0)	A pointer to the 6-byte ASCII field name for the current field. (An output value for MACRO-11 only.)
–	F$NUM(R0)	The field index for the current field (when that field is an indexed field). (An output value for MACRO-11 only.)

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
2	FS$INC	Current form incomplete.
–11	FE$FLD	Specified field does not exist (invalid field name or index).
–14	FE$NSC	Specified field not in scrolled area.
–17	FE$UTR	Undefined field terminator.
–19	FE$IFN	Specified call invalid in current context of form.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.17 FPUT – Output a Value to Specified Field

The Form Driver displays the value in the specified field and stores the value in the impure area. If the value to be displayed is shorter than the field for which it is intended, the Form Driver right or left justifies and zero or blank fills the field according to the definition of the field. If the value is longer than the field, the Form Driver truncates the displayed value, and if it contains Debug support, the Form Driver sets the status code to −16.

If the length of the value to be output is zero and the field has a default value, the Form Driver restores the default value to the screen and the impure area. If the field has no default value, the Form Driver clears the field.

The Form Driver does not validate either the specified or the default values.

BASIC-PLUS-2 and FORTRAN

```
CALL FPUT(fval,fid[,fidx])
```

COBOL

```
CALL "FPUT" USING BY DESCRIPTOR fval,
                  BY DESCRIPTOR fid[ ,BY REFERENCE fidx] .
```

DIBOL-83

```
XCALL FPUT(fval,fid[,fidx])
```

MACRO-11

```
$FDV ARG=arglst,FNC=PUT,REQ= reqlst,NAM=fld,NUM=idx,
        VAL=val,LEN=len
```

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
FID	NAM	A field name. For MACRO-11 only, a pointer to a 6-byte ASCII field name.
FIDX	NUM	A field index for the specified field (when the field is indexed).
-	LEN	The length of the data to be displayed (must be −1 for ASCIZ strings and 0 for restoring the default field value). (An input value for MACRO-11 only.)
FVAL	VAL	The field value to be displayed.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
−11	FE$FLD	Specified field does not exist (invalid field name or index).
−16	FE$DLN	Data specified for output too long (truncated by Form Driver).
−20	None	Wrong number of arguments in call. (For high-level language programs only.)
−21	None	Impure area not yet initialized. (For high-level language programs only.)

6.18 FPUTAL – Output Values to All Fields

The Form Driver outputs the specified data to all fields in the form. The data for each field must match the length of that field. Data must be arranged in the order in which the Form Driver would retrieve the fields if an FGETAL call were issued for the form.

If the data supplied is too long, the Form Driver returns an error to the task and truncates the data. If the data is too short, the Form Driver outputs the default for any fields for which no data is provided.

If the length of the specified data is zero, the Form Driver restores default values for all fields in the form and clears any fields that do not have defaults. (If the form contains any scrolled areas, the offset to the current line for each of those areas is reinitialized to zero, with the top line as the current line.) Thus, the FPUTAL call with data length of zero provides one method of reinitializing a form. It is the only form of the call valid for a form that contains a scrolled area.

The Form Driver does not validate data output to fields from the application task. This is true for both explicit output and default values.

BASIC-PLUS-2 and FORTRAN

CALL FPUTAL*(fval)*

CALL FPUTAL

COBOL

CALL "FPUTAL" USING BY DESCRIPTOR *fval.*

CALL "FPUTAL".

DIBOL-83

XCALL PUTAL*(fval)*

XCALL PUTAL

MACRO-11

$FDV ARG=*arglst,*FNC=PAL, REQ= *reqlst,*VAL=*val,*LEN=*len*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
–	LEN	The length of the data to be displayed (must be –1 for ASCIZ strings and 0 for restoring the default field values). (An input value for MACRO-11 only.)
FVAL	VAL	The concatenated field value(s) to be displayed.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–12	FE$NOF	No fields defined for current form.
–16	FE$DLN	Data specified for output too long (truncated by Form Driver).
–19	FE$IFN	Specified call invalid in current context of form.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.19 FPUTL – Output to Last Line of Screen

The Form Driver clears the last line of the screen and displays the specified string on that line. On the VT200 terminal with the advanced video option, the Form Driver always applies the bold attribute to the last line of the screen when data is displayed there with the FPUTL call. On other VT200 terminals, the line appears underlined or in reverse video matching the cursor the terminal is set to use. (The *VT200 User Guide* describes how to change the VT200 cursor.)

If the string is longer than the current maximum line length for the terminal (80 or 132 characters), the Form Driver sets the status code as shown below and truncates the string when displaying it. If the length of the string is zero, the Form Driver clears the last line.

This call provides the only means by which your task can access the last line of the screen. Otherwise, the last line is reserved for use by the Form Driver to display error messages and help text.

The Form Driver does not examine data output to the last line from the application task.

BASIC-PLUS-2 and FORTRAN

CALL FPUTL*(fval)*

CALL FPUTL

COBOL

CALL "FPUTL" USING BY DESCRIPTOR *fval.*

CALL "FPUTL".

DIBOL-83

XCALL FPUTL*(fval)*

XCALL FPUTL

MACRO-11

$FDV ARG=*arglst,*FNC=LST,REQ= *reqlst,*VAL=*val,*LEN=*len*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
-	LEN	The length of the data to be displayed (must be −1 for ASCIZ strings and 0 for clearing the last line of the screen). (An input value for MACRO-11 only.)
FVAL	VAL	The string to be displayed on the last line of the screen.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
−16	FE$DLN	Data specified for output too long (truncated by Form Driver).
−20	None	Wrong number of arguments in call. (For high-level language programs only.)
−21	None	Impure area not yet initialized. (For high-level language programs only.)

6.20 FRETAL – Return Values for All Fields

FRETAL returns a concatenated string of the current values for all fields in the form. The order of the fields is the same as for the FGETAL call.

BASIC-PLUS-2 and FORTRAN

CALL FRETAL*(fval)*

COBOL

CALL "FRETAL" USING BY DESCRIPTOR *fval.*

DIBOL-83

XCALL RETAL*(fval)*

MACRO-11

$FDV ARG=*arglst,*FNC=RAL,REQ= *reqlst*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
–	F$LEN(R0)	The total length of all fields in the form.
FVAL	F$VAL(R0)	The concatenated values for all fields in the form. For MACRO-11 only, a pointer to the concatenated fields in the impure area.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–12	FE$NOF	No fields defined for current form.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (BASIC-PLUS-2 only.)

6.21 FRETN – Return the Value for the Specified Field

The most common use of the FRETN call is to get the value of a particular field, after a call to get all fields (FGETAL). The FRETN call can be issued at any time after the Form Driver displays the form. The FRETN call always returns the current contents of a field.

By using the FGETAL and FRETN calls to complement each other, you avoid having your task deal with a buffer that contains all the operator responses for the form. You can still take advantage of the Form Driver's management of all terminal interaction and use the FRETN call to access one field at a time. Other calls can be issued between FGETAL and FRETN.

BASIC-PLUS-2 and FORTRAN

`CALL FRETN(`*fval,fid[,fidx]*`)`

COBOL

```
CALL "FRETN" USING BY DESCRIPTOR fval, BY DESCRIPTOR fid
                    [,BY REFERENCE fidx].
```

DIBOL-83

`XCALL FRETN(`*fval,fid[,fidx]*`)`

MACRO-11

`$FDV ARG=`*arglst*`,FNC=RTN,REQ=` *reqlst*`,NAM=`*fld*`,NUM=`*idx*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
FID	NAM	A field name. For MACRO-11 only, a pointer to a 6-byte ASCII field name.
FIDX	NUM	A field index for the specified field (when the field is indexed).
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.
–	F$LEN(R0)	The length of the field.
FVAL	F$VAL(R0)	The field value, including padding.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion
–11	FE$FLD	Specified field does not exist (invalid field name or index).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)
–22	None	Returned string is longer than the declared variable length. (For high-level language programs only.)

6.22 FSHOW – Display a Form

The Form Driver clears only the portion of the screen required for the specified form and, beginning at the starting line number, clears the screen and displays the form. When first displayed, the form includes all text and the default values for all fields. The Form Driver clears any fields for which defaults were not assigned with the Form Editor.

If a starting line number is not specified in the call, the Form Driver uses the starting line number that was assigned with the Form Editor.

If a first line is specified in the call, the Form Driver starts to display the form at that line. If the form description specifies that the entire screen is to be cleared (Lines 1 through 23), the Form Driver ignores the line number specified in the call.

BASIC-PLUS-2 and FORTRAN

CALL FSHOW(*fnam[,line]*)

COBOL

CALL "FSHOW" USING BY DESCRIPTOR *fnam*[,BY REFERENCE *line*].

DIBOL-83

XCALL FSHOW(*fnam[,line]*)

MACRO-11

$FDV ARG=*arglst*,FNC=SHO, REQ= *reqlst*,NAM=*form*,NUM=*line*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
-	ARG	A pointer to the Argument List.
-	REQ	A pointer to the Required Arguments List.
FNAM	NAM	A form name. For MACRO-11 only, a pointer to a 6-byte form name.
LINE	NUM	The explicit starting line number for the form, overriding the line number assigned with the Form Editor.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
– 6	FE$ICH	Invalid channel under number specified.
– 7	FE$FCH	Form library not open on specified channel.
– 8	FE$FRM	Invalid form definition.
– 9	FE$FNM	Specified form does not exist.
–10	FE$LIN	Invalid first line number to display form.
–18	FE$IOR	Error encountered reading form library (an FCS or RMS system error code that provides more detail can be found with the FSTAT call and is returned in the second word of the status block).
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.23 FSPOFF – Turn Supervisor-Only Mode Off

When supervisor-only mode is on (the default choice), the Form Driver treats fields with the Supervisor Only attribute as display-only. The operator cannot enter data in such fields. With the FSPOFF call, your program can turn supervisor-only mode off, making fields with the Supervisor Only attribute accessible to the operator.

BASIC-PLUS-2 and FORTRAN

```
CALL FSPOFF
```

COBOL

```
CALL "FSPOFF".
```

DIBOL-83

```
XCALL SPOFF
```

MACRO-11

```
$FDV ARG=arglst,FNC=SPF,REQ=reqlst,
```

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.24 FSPON – Turn Supervisor-Only Mode On

The FSPON call turns supervisor-only mode on (the original or default condition). Fields having the Supervisor Only attribute are handled by the Form Driver as display-only; the terminal operator cannot access them.

BASIC-PLUS-2 and FORTRAN

```
CALL FSPON
```

COBOL

```
CALL "FSPON".
```

DIBOL-83

```
XCALL FSPON
```

MACRO-11

$FDV ARG=*arglst*,FNC=SPN,REQ=*reqlst*

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
–	ARG	A pointer to the Argument List.
–	REQ	A pointer to the Required Arguments List.
Outputs		
None	None	The status code is set. For MACRO-11 only, R0 points to the Argument List.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

6.25 FSTAT – Return the Status From the Last Call

The FSTAT call returns the status from the last call to the Form Driver. The secondary status, STAT2, returns an RMS error code, depending on the version of the Form Driver you are using. This STAT2 value is useful only when the value of STATUS is –4 or –18, indicating a problem while opening or reading a form library file. The FCS error codes are documented in the *RSX-11 I/O Operations Manual.* The RMS error codes are documented in the *RMS-11 User's Guide.*

BASIC-PLUS-2 and FORTRAN

```
CALL FSTAT(stat[,stat2])
```

COBOL

```
CALL "FSTAT" USING BY REFERENCE stat[ ,BY REFERENCE stat2].
```

DIBOL-83

```
XCALL FSTAT(stat[,stat2])
```

Inputs and Outputs

High-Level Language Argument Abbreviation	MACRO-11 Keyword or Offset	Requirement or Value
Inputs		
None	–	
Outputs		
STATUS	–	A numeric code for the completion status of the last Form Driver call that was executed.
STAT2	–	A numeric RMS or FCS status code for detailed information when the status value is –4 or –18.

Returned Status Values and Codes

Status Value High-Level Languages	Status Code (MACRO-11)	Meaning
1	FS$SUC	Successful completion.
–20	None	Wrong number of arguments in call. (For high-level language programs only.)
–21	None	Impure area not yet initialized. (For high-level language programs only.)

Chapter 7
Form Driver Programming Techniques and Examples

This chapter discusses programming techniques for using the Form Driver. The techniques discussed include scrolling, simultaneous display of multiple forms, emulating the FGETAL call with FGET and FPFT, and using indexed fields. The last section contains examples of Form Driver programming techniques.

7.1 Scrolling Techniques

A scrolled area is defined as a number of consecutive lines with identical format. The purpose of the scrolled area is to allow entry, edit, and review of more data than can be displayed on the screen at one time. It is, in effect, a window into a data base managed by your calling task. The size of the data base is determined by your task, and size is not limited by the Form Driver's impure area.

Under the direction of your task, the Form Driver controls the scrolled area using a series of calls to get data and to process field terminators. The Form Driver maintains a current line in a scrolled area and restricts your task to accessing that line only. Thus, the Form Driver ignores the index value argument in a scrolled area.

When a form is displayed, the current line in each scrolled area is initialized to the top line. The current line's identity is updated when the Form Driver processes field terminators.

To provide complete support for scrolling, your task must analyze the field terminators to be processed and update the screen with the appropriate data from the data base. To do this, the task must maintain pointers into the data base to the current line and the current window of the scrolled area. With this information, your task can control the scrolled area and the position of the current line.

When lines are scrolled forward, the task must provide the data to be displayed on the bottom line of the scrolled area. When lines are scrolled backward, the task must provide the data to be displayed on the top line of the scrolled area . Therefore, your task must know the number of lines in the scrolled area and maintain pointers into the data base for the current top and bottom lines of your screen as the data is scrolled forward and backward.

To some extent, when a scrolled area is scrolled physically, it is under the control of the application program. If the current line of a scrolled area is the bottom line, the Form Driver will always scroll the area when the scroll forward terminator is processed. If the current line is not the bottom line, the area is scrolled only if the application program specifies data to update the bottom line in the Form Driver call to process the scroll forward terminator (FPFT). Otherwise, the cursor moves down one line and that line becomes the new current line.

The opposite applies to scroll backward. If the current line is the top line, scroll backward always causes the scrolled area to scroll. If the current line is any other line, the area scrolls only if data to update the top line is specified in the Form Driver call to process the scroll backward terminator. Otherwise, the cursor moves up one line, and that line becomes the new current line.

The Form Driver provides calls to get the current scrolled line (FINLN) and to output data to the current scrolled line (FOUTLN) to aid you in implementing support for scrolled areas. If you wish to validate fields within a scrolled area on an individual basis, you can use calls to get a specified field (FGET) in combination with the call to process a field terminator (FPFT) to handle input in a scrolled line.

The calls to get all fields (FGETAL) and to get any field (FGETAF) are illegal for a form that contains a scrolled area. The call to output data to all fields (FPUTAL) is legal for a form with a scrolled area only in the special case that restores the default values to all fields.

When you define forms containing scrolled areas, remember that the Form Driver does not maintain text within a scrolled area (other than field-marker characters) after the text scrolls off the screen.

If, however, you do not wish to support all the features of scrolling, you can opt to display error messages instead of processing certain field terminators.

7.2 Three Common Scrolling Methods

The three common methods described below for using scrolled areas do not exhaust the possibilities. You might think of other methods more suitable for your application.

7.2.1 Entry, Edit, and Review

The first method can be called entry, edit, and review. This method uses a scrolled area to gather many lines of data, each one of which contains several fields. In this case, operator interaction with the form is similar to operator interaction with an FGETAL call. The scrolled area acts as a window into a segment of the data being collected. The operator is free to move about the form arbitrarily and to change the input data. When finished, the operator presses the ENTER key to signal that the data is complete.

If this method is to be used, your task must establish a data array containing enough space for the maximum number of scrolled lines desired. Employing the calls to get a scrolled line (FINLN) and process a field terminator (FPFT), the task exchanges control with the Form Driver and saves the data line-by-line as it is entered. Whenever the Form Driver passes a scrolling field terminator back to the application, the task must take appropriate action to position itself in the array.

Your task can control on which line of the scrolled area the Form Driver accepts data by using the call to process a field terminator (FPFT). Data to be displayed can be passed on the top or bottom line of the scrolled area. If no data is passed, the Form Driver simply moves the cursor up or down one line, provided the cursor is not on the first or last line of the scrolled area.

While scrolling, it is possible to reach the boundary of the available data space. In this situation, your task might, without processing the field terminator, print a line of text on the last line of the screen informing the terminal operator of the situation. The task can then reissue the call to get a scrolled line (FINLN).

If you wish to initialize the scrolled area to values other than the defaults for the fields it contains, you can issue calls to output a scrolled line (FOUTLN) and to process a field terminator (FPFT) along with the scroll forward and scroll backward terminators. Each line is filled by the FOUTLN call, followed by the FPFT call scrolling forward with no data passed. In this way, each line is filled in turn. After the last line is filled, no scroll forward is performed. You can reposition the cursor to the desired line by issuing several FPFT calls with scroll backward terminators.

If you want to reinitialize all scrolled area lines on the screen with their default field values, you can use the call to put all fields (FPUTAL); this passes no data. The entire form is then reinitialized.

A form can have more than one scrolled area, or can be complex, because several nonscrolled fields are scattered on either side of the scrolled areas. In this situation, your task can emulate a get all fields operation using the following procedure: the task uses a field name of asterisk (*) to get the first field. Get-field (FGET) and process-field-terminator (FPFT) calls are then used until the field returned from the FPFT call is in a scrolled area. The method of scrolling through an area line-by-line can then be used until the terminator received is one of the exit-scrolled-area terminators. At that point, control returns to the previous loop, with the ENTER key terminator signaling that the form is complete.

7.2.2 Normal and Display Only Fields

The second programming method for scrolled areas employs scrolled lines containing both normal and Display Only fields. In this method, the application task uses calls to get a field (FGET) to accept data from the operator. The task then validates fields individually. Using data entered by the operator, the task also computes, or finds in a data base, new values for Display Only fields. These new values are then output to the form by means of the call to put a field (FPUT).

With this method, the task must build scrolled lines from the individual fields so that the call to process a field terminator (FPFT) can be used data can be output to the top and bottom lines of the area when scrolling takes place. (The algorithm employed for the actual scrolling is essentially the same as in the first method described above.)

7.2.3 Reviewing a Data List

A third method of scrolling is useful for reviewing a list of data. In this method, a task provides access to a long list of data or other information and allows the operator to review the data or read the information. It is not possible, however, to get a scrolled line (FINLN) in a scrolled area whose fields are all Display Only. To circumvent this difficulty, you can use a single character field with the No Echo attribute to obtain a field terminator, thus allowing the task to scroll lines of data. The cursor appears on the line in the No Echo field to mark the point of interest for the operator.

7.3 Simultaneous Display of Multiple Forms

You can, if you wish, display more than one form at a time. If you specify a line number other than zero as the beginning of a form and use FSHOW rather than FCLRSH, the Form Driver offsets the form dynamically at run-time to overlay the currently displayed form. Keep in mind, however, that unless you provide a separate impure area for each form on the screen, the Form Driver knows about only the last form displayed, and your task can reference only that form.

In designing an application to display multiple forms simultaneously, it is important to define the HELP forms correctly. When the Form Driver restores the screen after a HELP form has been displayed, only the form for which the Form Driver currently has the impure area pointer can be restored. If, for example, two forms were displayed and the HELP form cleared the entire screen, only one form would be restored. To avoid this, each HELP form should clear only the area of the screen cleared by the form with which it is associated (as defined by the first and last line numbers).

The screen refresh function CTRL/W redisplays only the current form. Any HELP or other forms that are also on the screen are not refreshed and are therefore erased from the screen.

7.3.1 Impure Areas

Separate impure areas are required only if several forms are to be accessed interchangeably by calls to the Form Driver to get or put fields. If your application program is going to display one form, complete the processing for it and then display the next form without erasing the first. Only one impure area is required as long as your application makes no attempt to access the first form again. In this manner, any number of forms can be displayed simultaneously using only one impure area.

When you use more than one impure area to display multiple forms simultaneously, the task can switch control freely between the two forms. The forms themselves and any HELP forms they call upon must be defined with line numbers that do not interfere with each other.

7.3.2 HELP Forms

Displaying multiple forms simultaneously is particularly useful for HELP forms. It is possible to define HELP forms so that when displayed they leave the current form intact on the screen. Only the portion of the screen specified in the HELP form definition is cleared. Thus, if the HELP form is defined to be displayed between lines 15 and 23, the Form Driver clears only that portion of the screen. The original form occupies the remainder of the screen. Note, however, that if the Form Driver displayed the initial form with a line offset, it will use the same number to offset the HELP form. If the HELP form does not fit on the screen after the offset is applied, the Form Driver returns an error to your task. If you define a HELP form or any other form in its form description as being displayed between lines 1 to 23, the Form Driver clears the entire screen and, ignoring the starting line number in a call, displays the form as defined. When a HELP form and a current form are displayed simultaneously, the Form Driver redisplays the current form completely when the terminal operator signals that he or she is finished with the HELP form. All or any part of the HELP form that does not overlap the current form remains on the screen.

7.4 Emulating the FGETAL Call by Combining the FGET and FPFT Calls

The principal advantage of using the FGETAL call to get all fields in a form (FGETAL) is that this causes the Form Driver to take charge of all input at the terminal. An FGETAL call returns to your task only when the terminal operator signals completion by pressing the ENTER or RETURN key. The disadvantage of FGETAL, however, is that your task cannot respond to and edit input on a field basis as it is entered by the operator.

You can, however, emulate an FGETAL call with a combination of the get field (FGET) and process-field-terminator (FPFT) calls. A similar emulation can be performed with the FGETAF (get any field) and FPFT calls. (See Section 7.7.1 for an example of this.)

The call to return all fields (FRETAL) can be used to obtain the entire record for processing after the terminal operator has entered the form.

7.5 Using the FGETAF Call

If a form contains a number of fields any one of which would be sufficient input for the form, the call to get any field (FGETAF) is particularly useful. For example, a form might contain fields for an account number and a name. Your task needs one of these to locate a customer's record in a data base. The call to get any field (FGETAF) allows the terminal operator to enter data in either of the two fields. The operator can choose the one for which information is available. The Form Driver returns only the data for the chosen field to the task.

In a form that contains a menu from which the operator is to select one item, FGETAF can be used to allow the operator to position the cursor in the chosen field and press the ENTER key.

7.6 Using Indexed Fields

Identical fields on consecutive lines of a form can be defined as indexed. Indexed fields make it possible to design smaller forms with smaller impure area requirements. If a form has many fields, indexing can be an important advantage.

Your task can reference any field in a form, whether indexed or not indexed, by specifying the field name and index value (the index is an integer from 1 to n, where n is the repeat count for the field). If the field is not indexed, the Form Driver ignores the index value.

The order in which the Form Driver moves through indexed fields depends on whether the fields are defined as a vertical or horizontal array. If the array is defined as vertical, the Form Driver moves down the form through each element of the array before moving to the field after the first element.

You can define multiple fields as a horizontal array if:

1. The first occurrence of each field is on the same line of the form.
2. The repeat count is the same for all fields.
3. There are no intervening fields on the first line of the array.

If a field defined as a horizontal array does not meet all the above requirements, it will default to a vertical array.

In a horizontal array, the Form Driver moves across each line of the array, through each field, before going on to the next line. Thus, horizontally indexed fields allow grouping of related fields together, not only in the order in which data is entered on the screen but also in the order of the data returned by a call to get all fields (FGETAL).

7.7 Examples of Programming Techniques

This section contains several code segments written in FORTRAN IV. The programming techniques are the same for all languages. The last part of this section is a series of MACRO-11 program examples.

7.7.1 Emulating FGETAL with FGET and FPFT

The FGET and FPFT calls can be used to emulate a call to get all fields yet still allow the calling task to validate responses immediately on entry, before proceeding to the next field in the form.

```
CALL FCLRSH (FORM)            ! Display the form

CALL FGET (RESP, TERM, "*")   ! Get first field in form
CALL FGCF (FIELD)             ! Get the name of the field
GOTO 2                        ! Validate response if
                              ! necessary


1        CALL FGET (RESP, TERM, FIELD) ! Get a field

2        .
         . Validate the user's response.
         . Following validation, the variable "ERRVAL" is zero
         . if the response is valid, non-zero if invalid.
         .

         IF (ERRVAL .NE. 0) GOTO 1    ! Get field again on error
         IF (TERM .EQ. 0) GOTO 10     ! Branch if terminator was
                                      ! "ENTER"
         CALL FPFT                    ! Else process field terminator
         CALL FGCF (FIELD)            ! Get name of field to get
         GOTO 1                       ! Get next field

10       CALL FRETAL (DATA)           ! Return responses for all
                                      ! fields
         .
         .
         .
```

7.7.2 Table Lookup

Get the response for a field and validate it against a table of valid responses. The list of valid responses is contained in the form as named data. (The data name is the same as the field name.)

As an example, consider the field 'MONTH' defined as picture AAA. The corresponding named data 'MONTH' contains the table of valid responses for the field in the form.

Note that the named data is returned as an ASCII string terminated with a null.

```
1       CALL FGET (VALUE, TERM, FIELD) ! Get the field

        CALL FLEN (LENGTH, FIELD)    ! Get the field length

        CALL FNDATA (FIELD, VALID)   ! Get corresponding named data

        PTR = 1                      ! Initialize index

2       DO 3 I = 1,LENGTH            ! Check for valid response
        IF (VALID(PTR+I-1) .EX. 0) GOTO 5 ! Error if not
        IF (VALID(PTR+I-1) .NE. VALUE(I)) GOTO 4
3       CONTINUE
        GOTO 6

4       PTR = PTR + LENGTH           ! Update index
        GOTO 2

5       CALL FPUTL ("Illegal response to field") ! Display error
                                     ! message
        GOTO 1                       ! Get field again

6       .
        .
        .
```

7.7.3 Form Linkage

Named data can be used to provide automatic form linkage independent of the application program.

```
         CALL FCLRSH ("FIRST")        ! Display form

                                   "
1        CALL FGET (RESP, TERM, "")  ! Get first field in form
         CALL FGCF (FIELD)            ! Get the name of the field
         GOTO 3                       ! Process response

2        CALL FGET (RESP, TERM, FIELD) ! Get a field

3        .
         . Process response
         .

         IF (ERRVAL .NE. 0) GOTO 2    ! Get field again on error
         IF (TERM .EQ. 0) GOTO 4      ! Branch if terminator "ENTER"
         CALL FPFT                    ! Else process field terminator
         CALL FGCF (FIELD)            ! Get name of field to get
         GOTO 2                       ! Get next field

4        IF (FNDATA ("NXTFRM", FORM) .LT. 0) GOTO 5 ! Get name of
                                      ! next form
         CALL FCLRSH (FORM)           ! Display it if there is one

5        STOP                         ! Else exit
```

7.7.4 Menus and Application Data

Named data can be used to facilitate development of menu driven applications and to store form-specific information.

As an example, consider the menu form named FIRST in Appendix B, Figure B-1. The named data contains the name of the appropriate form to display for each of the possible functions and the name of the corresponding file to be written.

```
        CALL FCLRSH ("MENU")          ! Display menu form


1       CALL FGET (RESP, TERM, "FIELD") ! Get response

        IF (FNDATA (RESP, FORM) .GT. 0) GOTO 2 ! Get corresponding
                                      ! named data

        CALL FPUTL ("Illegal choice") ! If none, invalid response
        GOTO 1                        ! Get the field again

2       DNAM = RESP(1:1)//"F"         ! Get named data name for file
        CALL FNDATA (DNAM, FILE)      ! Get name of corresponding
                                      ! file

        CALL FCLRSH (FORM)            ! Display form and process
          .
          .
          .
```

7.7.5 Initializing a Scrolled Area

This example illustrates how to initialize a five line scrolled area in a form.

```
      CALL FCLRSH ("FORM")          ! Display form

C
C Initialize the first line in the scrolled area
C

      CALL FOUTLN ("FIELD", A(1,1)) ! Initialize first line

C
C Now scroll forward and initialize the next line until the
C screen is initialized.
C

      DO 1 I = 2 TO 5
      CALL FPFT (8, "FIELD")        ! Scroll forward to next line
      CALL FOUTLN ("FIELD", A(1,I)) ! Initialize line
 1    CONTINUE

C
C Now move back to the first line of the scrolled area to
C solicit input.
C

      DO 2 I = 1 TO 4
      CALL FPFT (9, "FIELD")        ! Scroll backward to previous
                                    ! line
 2    CONTINUE

      .
      .
      .
```

7.7.6 MACRO-11 Programming Examples

This section contains MACRO-11 examples. Each of the following examples assumes that the initialization code precedes it.

```
          .MCALL  $FDV,$FDVDF,$EXIT
          $FDVDF                       ; DEFINE ARGUMENT LIST SIZES
                                       ; (F$ASIZ AND F$RSIZ)

          .
          .
          .

ISIZ      = 1024.                      ; IMPURE AREA SIZE IN BYTES
ARGLST:   .BLKB   F$ASIZ               ; ALLOCATE SPACE FOR ARGUMENT
                                       ; LIST
REQLST:   .BLKB   F$RSIZ               ; ALLOCATE SPACE FOR REQUIRED
                                       ; ARG LIST
IMPURE:   .WORD   ISIZ                 ; SIZE OF IMPURE AREA IN BYTES
                                       ; IN FIRST WORD
          .BLKB   ISIZ-2               ; THE IMPURE AREA
STAT:     .BLKW   2                    ; 2 WORD STATUS BLOCK REQUIRED

          .
          .
          .

; Initialize the required arguments list

          MOV     #REQLST,R0           ; REQUIRED ARGUMENTS LIST
                                       ; POINTER
          MOV     #STAT,F$STS(R0)      ; STATUS BLOCK POINTER
          MOVE    #1,F$CHN(R0)         ; LIBRARY CHANNEL NUMBER
          MOV     #IMPURE,F$IMP(R0)    ; IMPURE AREA POINTER
          $FDV    ARG=#ARGLST,REQ=#REQLST ; INIT REQ ARG LIST
                                       ; POINTER

          .
          .
          .
```

In each of the following examples, the Argument List pointer is specified in the first call to the Form Driver only. The pointer does not have to be specified in succeeding calls unless R0 (which contains the Argument List pointer) is modified.

The following examples also assume a debugged application. In such an application, the Form Driver should return no errors except for I/O errors resulting from reading forms. For this reason, the examples below check for errors only on return from calls to display forms or to solicit input from the terminal operator. An I/O error can result if the terminal operator requests a HELP form or screen refresh.

7.7.6.1 Example 1: FGET/FPFT in Place of FGETAL — This example emulates a call to get all fields by using successive calls to get a specified field and process a field terminator.

```
        .
        .
        .

FIELD1: .ASCII  /* /                    ; FIRST CHARACTER OF FIELD
                                        ; NAME '*'
                                        ; TO GET FIRST FIELD IN FORM
FORM:   .ASCII  /FORM /                 ; FORM NAME

        .
        .
        .
        $FDV    ARG=#ARGLST,FNC=CSH,NAM=#FORM,NUM=#0 ; DISPLAY
                                        ; FORM
        BCS     ERROR                   ; BR IF I/O ERROR

10$:    $FDV    NAM=#FIELD1             ; INITIALIZE ARGUMENT LIST FOR
                                        ; FIRST
                                        ; FIELD TO GET

; Get a specified field

20$:    $FDV    FNC=GET                 ; GET THE SPECIFIED FIELD
        BCS     ERROR                   ; BR IF I/O ERROR
        MOV     F$NAM(R0),R1            ; GET POINTER TO FIELD NAME
        CALL    PRSFLD                  ; CALL ROUTINE TO PROCESS
                                        ; FIELD
        BCC     30$                     ; ON RETURN C-CLR IF VALID
                                        ; INPUT
                                        ; ELSE R1 POINTS TO ASCIZ
                                        ; ERROR MESSAGE
; Invalid input

        $FDV    FNC=LST,VAL=R1,LEN=#-1
                                        ; OUTPUT ERROR MESSAGE TO
                                        ; LAST LINE
        BR      20$                     ; GET SAME FIELD AGAIN

; Valid input

30$:    $FDV    FNC=TRM                 ; PROCESS FIELD TERMINATOR
        CMP     #FT$NTR,F$TRM(R0)       ; FIELD TERMINATOR = ENTER?
        BNE     20$                     ; BR IF NOT
        CMP     #FS$INC,STAT            ; ELSE CHECK FOR INCOMPLETE
                                        ; FORM
        BEQ     20$                     ; GET FIELD IF FORM INCOMPLETE

; Done with form, reinitialize to get again

40$:    $FDV    FNC=PAL,LEN=#0          ; RESTORE DEFAULT VALUES TO
                                        ; ALL FIELDS
        BR      10$                     ; GET FORM AGAIN

        .
        .
        .
ERRMSG: .ASCIZ  *Fatal I/O Error*
        .EVEN
ERROR:  $FDV FNC=LST, VAL=#ERRMSG, LEN=#-1
        $EXIT
```

7.7.6.2 Example 2: Named Data — This example illustrates a possible use of the named data feature.

```
        .
        .
        .

MENU:   .ASCII  /MENU /              ; FORM NAME

SELERR: .ASCII  /ERROR - SELECT ANOTHER FORM/ ; ERROR MESSAGE
ERRLEN =        .-ERRMSG
        .EVEN

        .
        .
        .

; Display menu form - a form with a list of functions with a
; number from 1-5 associated with each function. The form
; contains only one field - a 1-byte numeric field. Therefore
; the data returned from a call to get all fields is
; guaranteed to be a 1-byte ASCII digit from 0-9.

        $FDV    ARG=#ARGLST,FNC=CSH,NAM=#MENU,NUM=#0 ; DISPLAY FORM
        BCS     ERROR                ; BR IF I/O ERROR

10$:    $FDV    FNC=ALL              ; GET ALL FIELDS
        BCS     ERROR                ; BR IF I/O ERROR
        MOVB    @F$VAL(R0),R1        ; GET 1 BYTE RESPONSE
        SUB     #'0,R1               ; CONVERT DECIMAL TO BINARY VALUE
```

```
; Get named data by number (using number entered in menu form)


        $FDV    FNC=DAT,NAM=#0,NUM=R1 ; GET NAMED DATA
        BCC     20$                   ; IF C-CLR, NAMED DATA FOUND
                                      ; ELSE INVALID NUMBER (INVALID
                                      ; INPUT)

; Invalid input in menu form

        $FDV    FNC=LST,VAL=#SELERR,LEN=#ERRLEN ; OUTPUT MESSAGE TO
                                      ; LAST LINE
        BR      10$                   ; GET FIELD AGAIN

; Valid input - display requested form

20$:    $FDV    FNC=CSH,NAM=F$VAL(R0),NUM=#0 ; DISPLAY FORM
BCS     ERROR   ; BR IF I/O ERROR

        .
        .
        .
ERRMSG: .ASCIZ  *Fatal I/O Error*
        .EVEN
ERROR:  $FDV FNC=LST, VAL=#ERRMSG, LEN=#-1
        $EXIT
```

7.7.6.3 Example 3: Combining the FGETAL and FRETN Calls — This example illustrates use of the get all fields and return field calls.

```
        .
        .
        .

FORM:   .ASCII  /FORM01/                ; FORM NAME

FLDLST: .ASCII  /FIELD1/                ; LIST OF FIELD NAMES
        .WORD   PRSF1                   ; AND ROUTINE TO PROCESS EACH
FIELD
        .ASCII  /FIELD2/
        .WORD   PRSF2
        .ASCII  /FIELD3/
        .WORD   PRSF3
        .ASCII  /FIELD4/
        .WORD   PRSF4
ENDLST = .

        .
        .
        .

        $FDV    ARG=#ARGLST,FNC=CSH,NAM=#FORM,NUM=#0 ; DISPLAY
                                        ; FORM
        BCS     ERROR                   ; BR IF I/O ERROR
        $FDV    FNC=ALL                 ; GET ALL FIELDS
        BCS     ERROR                   ; BR IF I/O ERROR
        MOV     #FLDLST,R1              ; GET POINTER TO FIELD LIST
10$:    $FDV    FNC=RTN,NAM=R1          ; GET RESPONSE FOR FIELD
        ADD     #6,R1                   ; SKIP OVER FIELD NAME
        CALL    @(R1)+                  ; CALL ROUTINE TO PROCESS FIELD
        CMP     R1,#ENDLST              ; ALL FIELDS DONE?
        BLo     10$                     ; REPEAT IF NOT

        .
        .
        .
ERRMSG: .ASCIZ  *Fatal I/O Error*
        .EVEN
ERROR:  $FDV FNC=LST, VAL=#ERRMSG,LEN=#-1
        $EXIT
```

7.7.6.4 Example 4: Using FGET to Synchronize with Terminal Operator — This example shows use of a "special get" call to synchronize the task with the terminal operator.

```
         .


         .
         .

FORM1:   .ASCII   /FORM01/             ; FORM 1
FORM2:   .ASCII   /FORM02/             ; FORM 2
FIELD:   .ASCII   /FLDNAM/             ; NAME OF DISPLAY-ONLY FIELD

         .
         .
         .

         $FDV     ARG=#ARGLST,FNC=CSH,NAM=#FORM1,NUM=#0 ; DISPLAY
                                       ; FIRST FORM
         BCS      ERROR                ; BR IF I/O ERROR
         $FDV     FNC=ALL              ; GET ALL FIELDS
         BCS      ERROR                ; BR IF I/O ERROR

         .
         . Process data (get pointer to data to output in R1,
         length in R2)
         .

         $FDV     FNC=PUT,NAM=#FIELD,VAL=R1,LEN=R2 ; OUTPUT TO DISPLAY

                                                   ; ONLY FIELD

         $FDV     FNC=GET,NAM=#0       ; DO GET WITH NO FIELD SPECIFIED

                                       ; TO WAIT FOR USER ACKNOWLEDGMENT


         BCS      ERROR                ; BR IF I/O ERROR

         $FDV     FNC=CSH,NAM=#FORM2   ; DISPLAY NEXT FORM
         BCS      ERROR                ; BR IF I/O ERROR

         .
         .
         .
ERRMSG:  .ASCIZ   *Fatal I/O Error*
         .EVEN
ERROR:   $FDV FNC=LST, VAL=#ERRMSG, LEN=#-1
         $EXIT
```

Chapter 8
Preparing Your System for FMS-11 Applications

This chapter describes how to install and prepare your system for the FMS-11 application program. The three main topics are:

1. RSX system generation options
2. FMS installation procedures
3. FMS configuration procedures

8.1 RSX System Generation Options

With RSX-11M and RSX-11M-PLUS systems, the Form Driver depends on terminal service and mapping features that you can select only when you perform the system generation procedure.

8.1.1 Terminal Service Option

On RSX-11M systems with V4.2 software, the full-duplex terminal driver is the driver that is built into your software system by default. You can run all FMS software with the full-duplex terminal driver, and the Form Editor requires that driver. The full-duplex terminal driver must include support for get-multiple characteristics and set-multiple characteristics. However, to provide support for FMS applications on unmapped RSX-11S and RSX-11M systems, you can also build the Form Driver to run with the older, half-duplex terminal driver. If the half-duplex terminal driver is used, it must provide support for unsolicited input character AST and transparent read and write.

8.1.2 Mapping Options

Most FMS users build and use mapped RSX-11M and RSX-11M-PLUS systems.

The advantages of mapped systems are:

1. Mapped systems provide a great deal more memory for applications than unmapped systems.
2. The FMS Form Editor and Form Utility require a mapped system.
3. Using one system for all FMS-related work is more convenient than using two systems for different parts of the work.

However, you can also build and use unmapped systems for your FMS applications. In general, the Form Driver modules and your programs will work properly. Because the Form Driver requires approximately 7500 bytes (decimal) of memory, you have to limit the size of your FMS program to the amount of memory remaining free for applications.

8.2 System Installation Procedures

To install all the FMS components, including the Form Driver modules, you have to execute the indirect command file FMSINS.CMD to:

1. Copy the distribution files from the distribution volumes to the system volumes.
2. Task build all FMS utilities.
3. Compile and task build the demonstration programs.

8.2.1 The RSX Procedure

To complete installation successfully, you must be logged in under a privileged account. If you are installing FMS-11 on an RSX-11M-PLUS system, your terminal must be set for MCR mode commands. It is assumed that FMS-11 is being installed on the system device. All FMS-11 files are moved into account [30,10]. A VT200 is required to run the Form Editor and the demonstration programs.

Approximately 3600 disk blocks are required to install FMS-11/RSX.

In the installation procedure below, the parameter 'ddn' is the name and unit number of the device on which FMS-11 is being installed. The parameter 'dev' is the name and unit number of the device on which the distribution media is mounted.

Installation Procedure

```
MCR>ASN ddn:=SY:                    ! Make the installation device
                                    ! the default system device
MCR>UFD ddn:[30,10]                 ! Create the UFD for FMS-11
MCR>SET /UIC=[30,10]                ! Set the default UIC
```

For magnetic tapes on RSX-11M systems:

```
MCR>ALL dev:                        ! Allocate the drive
MCR>FLX SY:=dev:FMSINS.CMD/DO       ! Copy installation command
                                    ! file
```

For magnetic tapes on RSX-11M-PLUS systems:

```
MCR>MOU dev:/FOR                    ! Mount tape as foreign device
MCR>FLX SY:=dev:FMSINS.CMD/DO       ! Copy installation command
                                    ! file
```

For disks on both systems:

```
MCR>MOU dev:FMSRSX               ! Mount the disk
MCR>PIP SY:/NV=dev:FMSINS.CMD    ! Copy installation command
                                 ! file
```

For all media:

```
MCR>@FMSINS                      ! Copy files from distribution
                                 ! media and build utilities
                                 ! and demo programs
```

The installation command file prompts for the distribution device. The FMS-11 files are copied to account [30,10] on the system device. The Form Editor (FED) and the Form Utility (FUT) are task built. The MACRO versions of the demonstration program are built automatically. The BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV, and FORTRAN-77 versions of the demonstration program are built only if the corresponding compilers are installed in the system. If the FORTRAN-77 compiler is installed in the system, it is assumed that the file LB:[1,1,]F4POTS.OLB exists as the FORTRAN-77 OTS. The installation command file asks the user if this is the case. If not, the FORTRAN-77 versions of the programs cannot be built.

8.3 Configuration Procedure for the Form Driver

The Form Driver has several features you might want for some of your applications, but not for all of them. For example, if you are using forms that include fixed-decimal fields, you must use a version of the Form Driver that provides fixed-decimal field support. Otherwise, since fixed-decimal field support increases the size of the Form Driver by about 200 words, you might want to use a smaller version of the Form Driver.

- Do you want to build the Form Driver without the SOB instruction?
- Do you want ONLY memory-resident form support?
- Do you want to delete fixed-decimal field support?
- Do you want to delete scrolled area support?
- Do you want debug error messages?
- Do you want support for the VT52 or VT100 terminal instead of the VT200?
- Size of the directory buffers in blocks?
- Number of directory buffers?
- Number of libraries open at a time?
- Do you want support for other than full-duplex terminal service? (RSX-11M/M-PLUS systems only)

To run the configuration procedure dialogue, the FMS software must be installed on your system. The following commands invoke the procedure.

For RSX-11M and RSX-11M-PLUS systems:

```
MCR>@FDVBLD RET
```

8.3.1 Question Types and Defaults

The configuration dialogue uses two types of questions:

1. Yes and No questions.
2. Questions that ask you to type a number.

For any question, you can ask for a short explanation of the question by typing the ESCAPE key in response to the question. After an explanation, the system displays the current question again.

Yes and No questions are in the following form:

```
> * Text of the question?  [Y/N]
```

You can respond as follows:

1. Type the ESCAPE key if you want an explanation.
2. Type Y and press the RETURN key for YES.
3. Type N and press the RETURN key for NO.
4. For the default response, press the RETURN key without typing another key. The default to all Yes and No questions is NO.

Questions that ask you to type a value are in the following form:

```
> * Text of the question [D R:x-y D:z]:
```

In the configuration dialogue, all numeric responses are in decimal. In the model above, the first letter D in the brackets stands for decimal. The second field within the brackets shows the range of valid answers. In the model above, R: stands for range, x stands for the smallest valid answer, and y stands for the largest valid answer. The third field within the brackets shows the default value. In the model above, D: stands for default, and z stands for the default value.

You can respond as follows:

1. Type the ESCAPE key if you want an explanation.
2. Type a number within the specified range and press the RETURN key.
3. For the default response, press the RETURN key without typing another key.

The following sections explain each of the configuration dialogue questions. Each section begins by quoting the short explanation the system displays if you type the Escape key in response to the question.

8.3.2 Do You Want to Build the Form Driver without the SOB Instruction? [Y/N]:

```
"Normally the Form Driver is  built to take advantage of the
hardware SOB instruction.  However, for those PDP-11 processors
that do not support the SOB instruction, the Form Driver can be
built to use a macro instead (adding about 50 words to its
size)."
```

8.3.3 Do You Want ONLY Memory-Resident Form Support? [Y/N]:

```
"Normally, the Form Driver has support for both memory-resident
and media-resident forms.  For an application running under
RSX-11S or which for other reasons must run in minimal memory,
select ONLY memory-resident form support to save space."
```

Section 5.4 explains how to include memory-resident forms with your programs.

8.3.4 Do You Want to Delete Fixed-Decimal Field Support? [Y/N]:

```
"If memory space is very critical and fixed-decimal fields are
not needed, about 200 words can be saved by deleting support for
fixed-decimal fields."
```

Chapter 2 explains fixed-decimal fields. If you do not know whether fixed-decimal fields appear in any of the forms you are using, use the Form Utility to get a printable description of each form and check each field description. Chapter 3 describes how to use the Form Utility.

8.3.5 Do You Want to Delete Scrolled Area Support? [Y/N]:

```
"If memory space is very critical and scrolled areas are not
needed, about 500 words can be saved by deleting support for
scrolled areas."
```

Chapter 2 describes how to create a scrolled area within a form, and Chapter 7 illustrates how to use scrolled areas. If you do not know whether any of the forms you are using contain scrolled areas, use the Form Utility to get a printable description of each form and check each field description. Chapter 3 describes how to use the Form Utility.

8.3.6 Do You Want Debug Error Messages? [Y/N]:

```
"While debugging an application, it is very helpful to have the
Form Driver signal errors in the application code at the
terminal.  Once an application is debugged, a Form Driver without
this support should be used to save space and to avoid having
these messages appear to operators."
```

Chapter 5 describes the debug error message features. Appendix E includes all of the messages.

8.3.7 Do You Want Support for the VT52? [Y/N]:

```
"The Form Driver was designed to take advantage of the features
of the VT200 terminal.  However, it is possible to build a Form
Driver to support the VT52 terminal.  A single Form Driver
library can support either the VT200 or the VT52 but not both.
In order to support both types of terminals, you must maintain
two Form Driver libraries."
```

8.3.8 Size of the Directory Buffers in Blocks [D R:1.-2.D:1.]:

```
"One-block buffers provide for form libraries with about 60
forms.  Two-block buffers provide for form libraries with about
120 forms."
```

Form library files and the directory buffers they require are needed only when you are using media-resident forms. Therefore, if you select only memory-resident forms in the first configuration dialogue question, the system does not ask you this question or the questions described in the next two sections.

The size to use for directory buffers depends on several factors. A one-block directory buffer is large enough for a 60-form (decimal) form library and a two-block buffer is large enough for a 124-form (decimal) form library file. Each form library file directory must reside in one directory buffer. When a directory is larger than the buffer size you have specified, only the forms for the part of the directory that fits can be accessed. When the Form Driver attempts to access a form for which there is no directory entry, the Form Driver signals that it cannot find the form.

The size of the directory buffer is also the size of the buffer for reading forms. Larger buffers allow faster access to forms because fewer I/O requests are required to read the form description. Space can be saved by allocating smaller buffers. The default directory buffer size is one block.

8.3.9 Number of Directory Buffers [D R:1.-20. D:1.]:

```
"The time required to access form directories can be optimized by
keeping directories for more form libraries in memory at the
expense of memory space.  Only one directory buffer is ever
required.   Buffers are made available on a least-recently-used
basis."
```

The number of directory buffers depends on several factors. Space can be reduced by allocating fewer buffers, but time to access form libraries is increased. With only one directory buffer, you can use any number of form library files simultaneously. The directories are read and reread as needed. Time to access form libraries is decreased by allocating more directory buffers, but the amount of space used is increased.

When you are using more form library files than the number of directory buffers you have allocated, the directory buffers are reallocated on a least-recently-used basis to optimize the directory buffer usage. When RMS support is selected, sufficient pool space is allocated for simultaneous use of as many form library files as you have specified. The following section describes how to specify the number of form library files that you will be using simultaneously. A one block buffer is also allocated for the open code of RMS to use.

8.3.10 Number of Libraries Open at a Time [D R:1.-20. D:1.]:

```
"Select the maximum number of form libraries which must be open
at the same time."
```

Successive libraries are opened by changing the channel to the required argument block. This channel corresponds to the logical unit number (LUN) that is assigned by the task.

8.3.11 Do You Want Support for Other than Full-Duplex Terminal Service? [Y/N] (RSX-11M/M-PLUS systems only):

```
"If the application is to run under RSX-11S or in an unmapped
RSX-11M system or is very space critical, memory can be saved by
not using the full-duplex terminal driver.  If the application is
using another terminal driver, the Form Driver must be
reconfigured to operate.  The terminal driver used must have
support for unsolicited input ASTs.''
```

The *RSX-11M/M-PLUS I/O Drivers Reference Manual* describes the full-duplex and half-duplex terminal drivers in detail.

For all systems:

After you have answered this question, the system displays the following notice.

```
"There are no more questions.  It will take a few minutes to
assemble the Form Driver and build the object libraries."
```

At the end of the configuration procedure, the system produces the following file in [30,10].

For RSX-11M/M-PLUS systems:

- FMSMAC.MLB, the FMS macro library.

For VT100 support:

- FDVLIB.OLB, the Form Driver library for applications that use FCS support.
- FDVLRM.OLB, the Form Driver library for applications that use RMS support.

For VT52 support:

- F52LIB.OLB, the Form Driver library for applications that use FCS support.
- F52LRM.OLB, the Form Driver library for applications that use RMS support.

8.4 Building and Running Your Application Tasks

8.4.1 Building and Running Application Programs

Application programs written in supported languages are built using the Task Builder in the normal way. In this chapter, the separate sections for each supported language include examples of task build procedures.

The high-level language interface object modules supplied are:

- HLLBP2.OBJ the interface for BASIC-PLUS-2.
- HLLCOB.OBJ the interface for COBOL-11 and COBOL-81.
- HLLDBL.OBJ the interface for DIBOL-83.
- HLLFOR.OBJ the interface for FORTRAN IV and FORTRAN-77.

8.4.2 Considerations When Using ODL

A few considerations must be followed when you include the Form Driver in an ODL structure.

The data module must always be in the root. To put it there, use the following module specification.

For RSX systems:

```
FDVLIB/LB:FDVDAT
```

With the full-duplex terminal driver, the Form Driver can be placed on any branch of the ODL by using a module specification.

For RSX systems:

```
FDVLIB/LB:FDV-FDVLIB/LB
```

With the full-duplex terminal driver, which can be used in applications running under RSX-11S, the following Form Driver factor must be included in the root.

For RSX systems:

```
FDVLIB/LB:FDVTIO
```

The language interface modules should be placed with the Form Driver library by using the following specification.

For RSX systems:

```
FORM:   .FCTR HLLDFN-FDVLIB/LB:FDV-FDVLIB/LB
```

Appendix A
FMS System MACRO Library

```
.NLIST
         .ENABLE LC
.LIST
.TITLE   FMSMAC - FMS Macro Library
.SBTTL   ************************************
.SBTTL   *                                  *
.SBTTL   *              FMSMAC      *
.SBTTL   *         .IDENT  /V02.0A/         *
.SBTTL   *                                  *
.SBTTL   ************************************
.SBTTL
.IDENT   /V02.00 /
;
;
;                          COPYRIGHT (C) 1985 BY
;               DIGITAL EQUIPMENT CORPORATION, MAYNARD, MA
;
;
; MODULE:          FMSMAC.MAC
;
; Version:         V02.00
;
; MODIFIED BY:
;
;
;+
; $ARTS
;
;         Issues a .MCALL for other commonly used MACROS.
;-
.MACRO   $ARTS
         .MCALL  $SAV20, $SAV30, $SAV50, BSECT
         .MCALL  ORIGIN, PSECT,  FSYM$,  NEGCOD
```

```
          .MCALL  $TTYIN, $TTINR, $PRINT, $TTFLS
          .MCALL  $TTYOUT, OFFSET, SETSYM
.IIF      NDF,NEWC$P, NEWC$P = 0                   ;CAV
.IIF      NE,NEWC$P, .MCALL SOB                    ;CAV
.ENDM     $ARTS
;+
; $SAV20
;
;         SAVE REGISTERS 2-0
;-
.MACRO    $SAV20
          JSR     R2,$SAV20
.ENDM     $SAV20
;+
; $SAV30
;
;         SAVE REGISTERS 3-0
;-
.MACRO    $SAV30
          JSR R3,$SAV30
.ENDM     $SAV30
;+
; $SAV50
;
;         SAVE REGISTERS R5-R0
;
;-
.MACRO    $SAV50
          JSR     R5,$SAV50
.ENDM     $SAV50
;+
; SOB
;
;         A software substitute for the SOB instruction for old PDP-11s.
;-
.MACRO    SOB R,A
          DEC R
          BNE A
.ENDM     SOB
;++
; Documentation of use of ORIGIN and PSECT
;
```

```
;         ORIGIN Section_name,List_of_Attributes
;
; This defines the default .PSECT
;
;         PSECT Section_name,List_of_Attributes
;
; This changes to a .PSECT (perhaps unnecessary)
;
;         PSECT *
;
; This reverts back to the last .PSECT referenced with
; the ORIGIN macro.
;--
;+
; UNORG
;
;         For internal use only.  Normally defined by ORIGIN and
;         invoked by PSECT.
;-
.MACRO   UNORG
.ENDM    UNORG
;+
; ORIGIN
;
;         Defines the PSECT to revert back to with PSECT *
;-
.MACRO   ORIGIN SECT,LIST
.MACRO   UNORG
.LIST    BEX
         PSECT SECT,<LIST>
.ENDM    UNORG
         PSECT <SECT>,<LIST>
.NLIST   BEX
.ENDM    ORIGIN
;+
; PSECT
;
;         Changes to a specified .PSECT or reverts back to
;         the one defined by ORIGIN when PSECT * is used.
;-
.MACRO   PSECT   SECT,LIST
.LIST    BEX
.IF      IDN     SECT,<*>
         UNORG
.NLIST   BEX
         .MEXIT
.IFF
```

```
   .IF NB <LIST>
 .PSECT   SECT,LIST
   .IFF
 .PSECT   SECT
 .ENDC
 .ENDC
 .NLIST   BEX
 .ENDM   PSECT
 ;
 ;+
 ; $ERROR
 ;
 ;        Print error message.
 ;-
 ;
 ;.MACRO  $ERROR   LVL,TXT
 ;        .GLOBL   $ARTER
 ;        PSECT   .TEXT.,<D>
 ;..A     =        .
 ;        .ASCIZ  \'LVL'-'TXT'\
 ;        PSECT   *
 ;        JSR     R4,$ARTER
 ;          .WORD ..A
 ;.ENDM   $ERROR
 ;+
 ; BSECT
 ;
 ;        Define bit origin for bit definitions.  This MACRO is
 ;        normally used in conjunction with the BS MACRO.
 ;-
 ;+
 ; BS
 ;
 ;        Defines the specified symbol as equivalent to the current
 ;        bit position and increments the bit counter to the next
 ;        position.
 ;-
.MACRO   BSECT     SECTNM,ARG,INIT
.MACRO   BS        ARG1,GBL
.IF      NB        <ARG1>
  .IF    B         <GBL>
         .LIST
         ARG1 = SECTNM
         .NLIST
  .IFF
         .LIST
         ARG1 == SECTNM
         .NLIST
  .ENDC
.ENDC
```

```
          SECTNM = SECTNM * 2
.ENDM     BS
.IIF      NDF,SECTNM, SECTNM = 0
.IIF      NB       <INIT>, SECTNM = 1
.IF       NB       <ARG>
  .IF       NE       <ARG>
            .REPT ARG
            BS
            .ENDR
  .ENDC
.ENDC
.ENDM     BSECT
;+
; NEGCOD
;
;         Define negative codes for symbols.  Normally used
;         in conjunction with the NC macro.
;-
;+
; NC
;         Defines a specified symbol as equivalent to the
;         current value of the section name.
;-
.MACRO    NEGCOD  SECTNM,ARG,INIT
.MACRO    NC      ARG1,GBL
.IF       NB,<ARG1>
  .IF     B,<GBL>
          .LIST
          ARG1 = SECTNM
          .NLIST
  .IFF
          .LIST
          ARG1 == SECTNM
          .NLIST
  .ENDC
.ENDC
SECTNM    = SECTNM - 1
.ENDM     NC

.IIF      NDF,SECTNM, SECTNM = 0
.IIF      NB,<INIT>, SECTNM = -1
.IIF      NB,<ARG>, SECTNM = -<ARG>

.ENDM     NEGCOD
;+
; FSYM$
;
;         This macro defines some form definition symbols.
```

```
;-

.MACRO   FSYM$
TE$XT    =        -2
FI$ELD   =        -3
NA$MED   =        -4
EF$ORM   =        -5
.ENDM    FSYM$


;+
; $TTFLS
;
;        Clear the input ring buffer of characters.
;-

.MACRO   $TTFLS
         CALL     $FMSFL
.ENDM    $TTFLS
;+
; $PRINT
;
;        Print a string of text.
;-
.MACRO   $PRINT   ADDR
         .GLOBL   $FMSPR
.IF      NB       <ADDR>
.IF      DIF      <ADDR>,R0
         MOV      ADDR,R0
.ENDC
.ENDC
         CALL     $FMSPR
.ENDM    $PRINT


;+
; $TTYIN
;
;        Input a character from the terminal.
;-

.MACRO   $TTYIN   CHAR
         CALL     $CHRIN
.IF      NB       <CHAR>
.IF      DIF      <CHAR>,R0
         MOVB R0,CHAR
.ENDC
.ENDC
.ENDM    $TTYIN
;+
; $TTINR
;
;         Input a character from the terminal without waiting
;-

.MACRO   $TTINR  CHAR
         CALL    $TTINR
.IF      NB      <CHAR>
.IF      DIF     <CHAR>,R0
         MOVB R0,CHAR
.ENDC
.ENDC
.ENDM    $TTINR

;+
; $TTYOUT
;
;         Output a character to the terminal.
;-

.MACRO   $TTYOUT  CHAR
.IF      NB       <CHAR>
.IF      DIF      <CHAR>,R0
         MOVB     CHAR,R0
.ENDC
.ENDC
         CALL     $CHROU
.ENDM    $TTYOUT
;+
;        FORM DRIVER CALL MACRO
;
;        $FDV ARGLST,FNC,REQ,NAM,NUM,TRM,VAL,LEN
;
; ARGLST OPTIONAL POINTER TO ARGUMENT LIST
; FNC             REQUIRED COMMAND FUNCTION CODE (LAST 3 LETTERS)
; REQ - LEN       OPTIONAL KEYWORD ARGUMENTS TO SUPPLY ARGUMENTS
;-
.MACRO   $FDV ARG,FNC,REQ,NAM,NUM,TRM,VAL,LEN
.MACRO   $$FA KWD,THG
.IF NB,THG
         MOV THG,KWD(R0)
.ENDC
.ENDM
.IF NB,ARG
         MOV ARG,R0
.ENDC
$$FA F$REQ,<REQ>
$$FA F$NAM,<NAM>
$$FA F$NUM,<NUM>
$$FA F$TRM,<TRM>
$$FA F$VAL,<VAL>
```

```
$$FA F$LEN,<LEN>
.IF NB,FNC
        MOV #FC$'FNC,F$FNC(R0)
        CALL $FDV
.ENDC
.ENDM

;       DEFINE THE SIZES OF THE ARGUMENT BLOCKS FOR FORM DRIVER
        .MACRO $FDVDF
        F$ASIZ = 16
        F$RSIZ = 10     ; F$STS , F$CHN , F$IMP , F$TCB
        F$TSIZ = 16     ; TCB block is 16 bytes long
.ENDM

;+
; $CSPON
;
;       This macro call CNSPON to put the terminal in
;       special mode and lower case.
;-

.MACRO  $CSPON
.ENDM   $CSPON

;+
; $CSPOF
;
;       This macro calls CNSPOF to put the terminal in
;       normal mode and upper case.
;-

.MACRO  $CSPOF
.ENDM  $CSPOF
;+
;    SETSYM and OFFSET are used to describe
;    structures without allocating storage.
;
;    Offset is called with the SYMBOL name, size
;    (in bytes) of  the storage the symbol represents,
;       and a initial value.
;
;    Contiguous calls to OFFSET result in automatic
;    increment of symbols. If a size is not specified,
;       2 bytes is the default.
;-
.MACRO SETSYM SYMB,NEXT,LCL
.IF  B,<LCL>
.LIST
SYMB == 'NEXT
.NLIST
```

```
.IFF
.LIST
SYMB = 'NEXT
.NLIST
.ENDC
.ENDM SETSYM

.MACRO OFFSET SYMBOL,STEP,INIT,LCL
.IF  NB,<INIT>
NXT = INIT
.ENDC
.IF  NDF,NXT
NXT = 0
.ENDC
SETSYM SYMBOL,\NXT,LCL
.IF  NB,<STEP>
INCR = STEP
.IFF
INCR = 2
.ENDC
NXT = NXT + INCR
.ENDM OFFSET
.END    ;End of FMSMAC.MAC
```

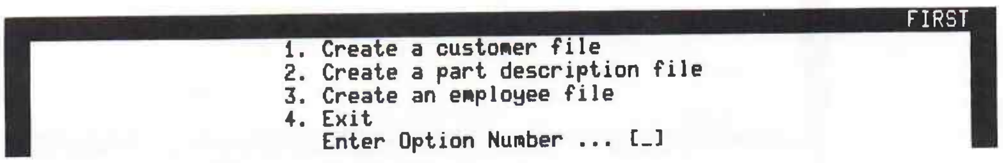
Appendix B
FMS Sample Forms

This appendix contains seven sample forms used by the demonstration programs included in this package.

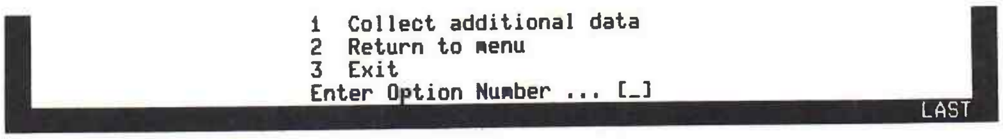
As an example of how the application uses the forms, the following sketch traces the major processing steps in creating a customer file.

Starting the Process

Initially, each version of the application displays a menu form named 'FIRST' to ask what process is to be done. Figure B-1 shows the form named 'FIRST' and the named data associated with that form.



```
                                                        FIRST
                    1. Create a customer file
                    2. Create a part description file
                    3. Create an employee file
                    4. Exit
                       Enter Option Number ... [_]
```



```
                      1  Collect additional data
                      2  Return to menu
                      3  Exit
                      Enter Option Number ... [_]
                                                        LAST
```

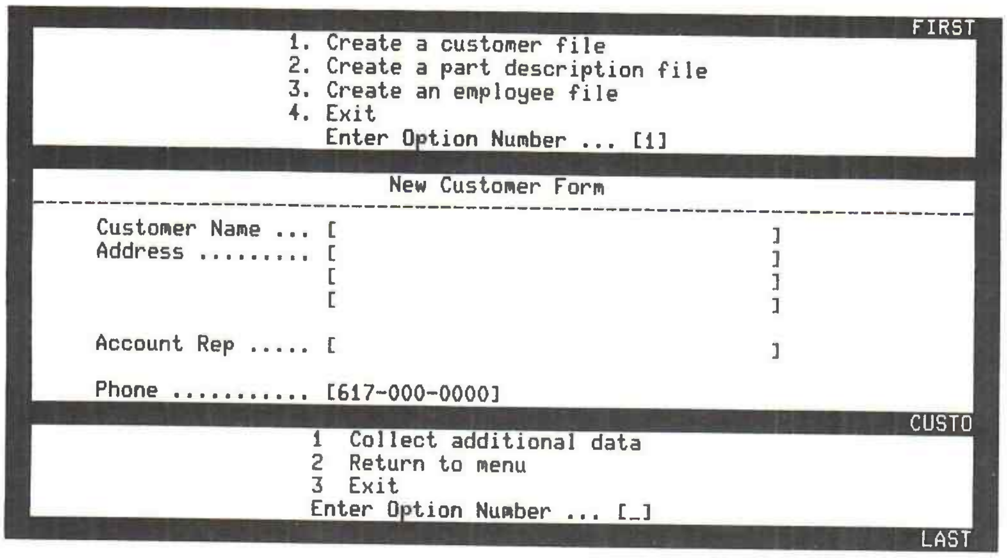
```
------------------------------------------------------------------------
!Named Data Information       !   Form name:          FIRST          !
!                             !   Help form name:                     !
! Name   Data                 !   First line:         1               !
!   1    CUSTO                !   Last line:          6               !
!   2    PARTS                !   Date created:        7-AUG-85       !
!   3    EMPLOY               !   Owner ID:           0               !
!   4    .EXIT.               !   Form attributes:    None            !
!  FILE1 SY:NEWCUS.DAT        !   Form length:        540 bytes       !
!  FILE2 SY:PARTS.DAT         !   Number of fields:   1               !
!  FILE3 SY:EMPLOY.DAT        !   Impure area size:   399 bytes       !
------------------------------------------------------------------------
```

Figure B-1. FIRST Form and Named Data Information for FIRST Form

The operator creates a customer file by typing '1' in response to the field prompt "Enter Option Number" in the menu form.

By checking the named data associated with the menu form, the application uses the operator's response to decide what form to display next. The application finds that the string 'CUSTO' corresponds to the operator's response in this case. The application then creates a new named data reference ('FILE1') from the operator's response, checks for it in the menu form's named data, and finds the corresponding string 'SY:NEWCUS.DAT.'

The application continues processing by loading and displaying the form 'CUSTO' in DEMLIB.FLB. (Figure B-2 shows the 'CUSTO' form and the named data associated with it.) The application also opens a customer data output file named NEWCUS.DAT on the system volume.



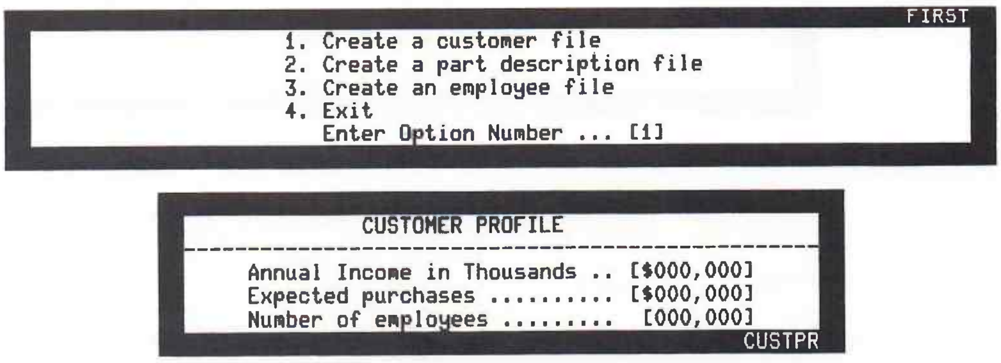
```
                                                                    FIRST
                          1. Create a customer file
                          2. Create a part description file
                          3. Create an employee file
                          4. Exit
                             Enter Option Number ... [1]

                               New Customer Form
------------------------------------------------------------------------
     Customer Name ... [                                  ]
     Address ......... [                                  ]
                       [                                  ]
                       [                                  ]

     Account Rep ..... [                                  ]

     Phone ........... [617-000-0000]
                                                                    CUSTO
                          1  Collect additional data
                          2  Return to menu
                          3  Exit
                          Enter Option Number ... [_]
                                                                     LAST
```

```
-----------------------------------------------------------------------
! Named Data Information       !  Form name:          CUSTO           !
!                              !  Help form name:                     !
!                              !  First line:         7               !
!   NXTFRM       CUSTPR        !  Last line:          18              !
!                              !  Date created:       7-AUG-85        !
!                              !  Owner ID:           0               !
!                              !  Form attributes:    None            !
!                              !  Form length:        924 bytes       !
!                              !  Number of fields:   7               !
!                              !  Impure area size:   612 bytes       !
-----------------------------------------------------------------------
```

Figure B-2. New Customer (CUSTO) Form and Named Data for New Customer Form

When the operator has completed the form named 'CUSTO,' the application writes the customer data to the output file. The application then uses the named data reference 'NXTFRM' to check for another form that is part of the customer data process. In the named data associated with the form named 'CUSTO,' the string 'CUSTPR' is associated with 'NXTFRM.'

The application continues processing by loading and displaying the form named 'CUSTPR.' Figure B-3 shows the 'CUSTPR' form and the named data associated with it. The file NEWCUS.DAT remains open.



```
                                                                    FIRST
                    1. Create a customer file
                    2. Create a part description file
                    3. Create an employee file
                    4. Exit
                       Enter Option Number ... [1]
```

```
                   CUSTOMER PROFILE
-------------------------------------------------------
     Annual Income in Thousands .. [$000,000]
     Expected purchases .......... [$000,000]
     Number of employees .........  [000,000]
                                                 CUSTPR
```

```
----------------------------------------------------------------------
! Named Data Information        !   Form name:          CUSTPR       !
!                               !   Help form name:                  !
!  Name      Data               !   First line:         7            !
! NXTFRM     .NONE.             !   Last line:          18           !
!                               !   Date created:       7-AUG-85     !
!                               !   Owner ID:           0            !
!                               !   Form attributes:    None         !
!                               !   Form length:        726 bytes    !
!                               !   Number of fields:   3            !
!                               !   Impure area size:   344 bytes    !
----------------------------------------------------------------------
```

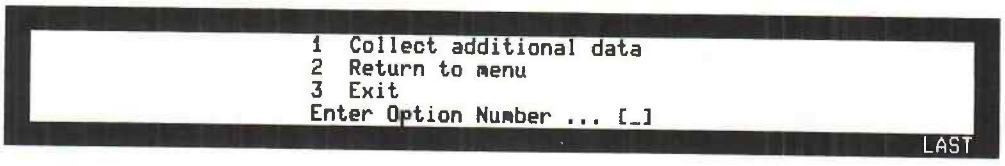
Figure B-3. Customer Profile Form and Named Data for Customer Profile Form

When the operator has completed the form named 'CUSTPR,' the application adds the responses to the customer data output file and checks the current named data for another form name associated with the reference 'NXTFRM.' The application gets the string '.NONE.'

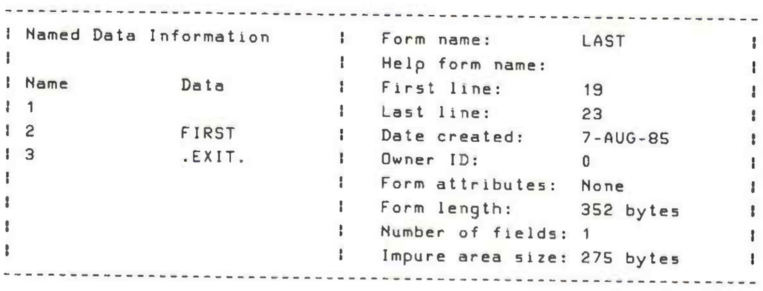
The application program and forms have been prepared so that the program gets the string '.NONE.' after completion of the last form in each data-entry process. The application then displays another menu form named 'LAST.'

Figure B-4 shows that form and the named data associated with it. The file NEWCUS.DAT remains open until the operator switches to another data-entry process or stops the application.

In the form 'LAST,' the default response in the field header is '1,' indicating that the operator wants to enter more customer data. The operator can type a different response and either stop the application or return to the first menu form ('FIRST') and choose another data entry process.



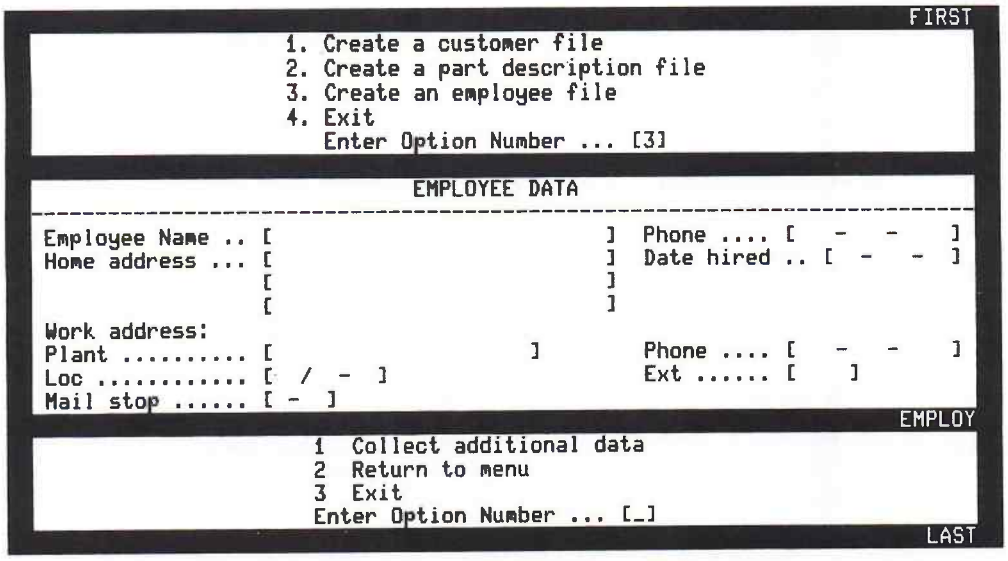
```
1  Collect additional data
2  Return to menu
3  Exit
Enter Option Number ... [_]
                                                        LAST
```



```
---------------------------------------------------------------------
! Named Data Information        !   Form name:          LAST          !
!                               !   Help form name:                   !
! Name           Data           !   First line:         19            !
! 1                             !   Last line:          23            !
! 2              FIRST          !   Date created:       7-AUG-85      !
! 3              .EXIT.         !   Owner ID:           0             !
!                               !   Form attributes:    None          !
!                               !   Form length:        352 bytes     !
!                               !   Number of fields:   1             !
!                               !   Impure area size:   275 bytes     !
---------------------------------------------------------------------
```

Figure B-4. LAST Form and Named Data for LAST Form

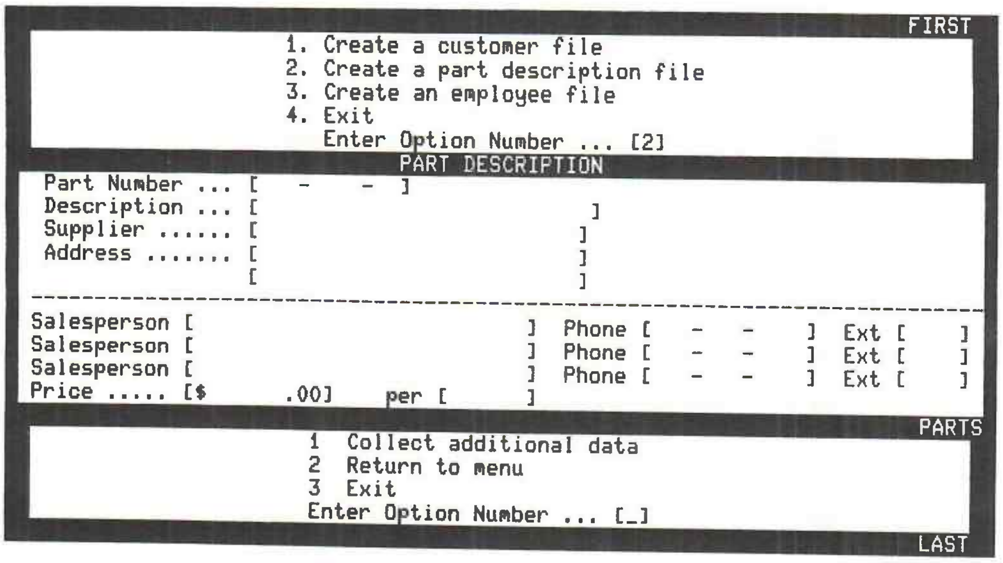
Figures B-5, B-6, and B-7 show the other forms in the form library file DEMLIB.FLB and the named data associated with each form. Refer to those figures as you trace the data-entry processes for creating a part description file and an employee file.



```
                                                                          FIRST
                         1. Create a customer file
                         2. Create a part description file
                         3. Create an employee file
                         4. Exit
                            Enter Option Number ... [3]

                               EMPLOYEE DATA
--------------------------------------------------------------------------------
 Employee Name .. [                           ]  Phone .... [   -   -   ]
 Home address ... [                           ]  Date hired .. [  -   -  ]
                  [                           ]
                  [                           ]
 Work address:
 Plant .......... [                     ]        Phone .... [   -   -   ]
 Loc ............ [  /  -  ]                     Ext ...... [    ]
 Mail stop ...... [ -  ]
                                                                         EMPLOY
                           1  Collect additional data
                           2  Return to menu
                           3  Exit
                           Enter Option Number ... [_]
                                                                           LAST
```

```
---------------------------------------------------------------------------
! Named Data Information          !  Form name:          EMPLOY        !
!                                 !  Help form name:                   !
! Name           Data             !  First line:         7             !
! NXTFRM         .NONE.           !  Last line:          18            !
!                                 !  Date created:       12-AUG-85     !
!                                 !  Owner ID:           0             !
!                                 !  Form attributes:    None          !
!                                 !  Form length:        1144 bytes    !
!                                 !  Number of fields:   11            !
!                                 !  Impure area size:   754 bytes     !
---------------------------------------------------------------------------
```

Figure B-5. Employee Data Form (EMPLOY) and Named Data Information for Employee Data form

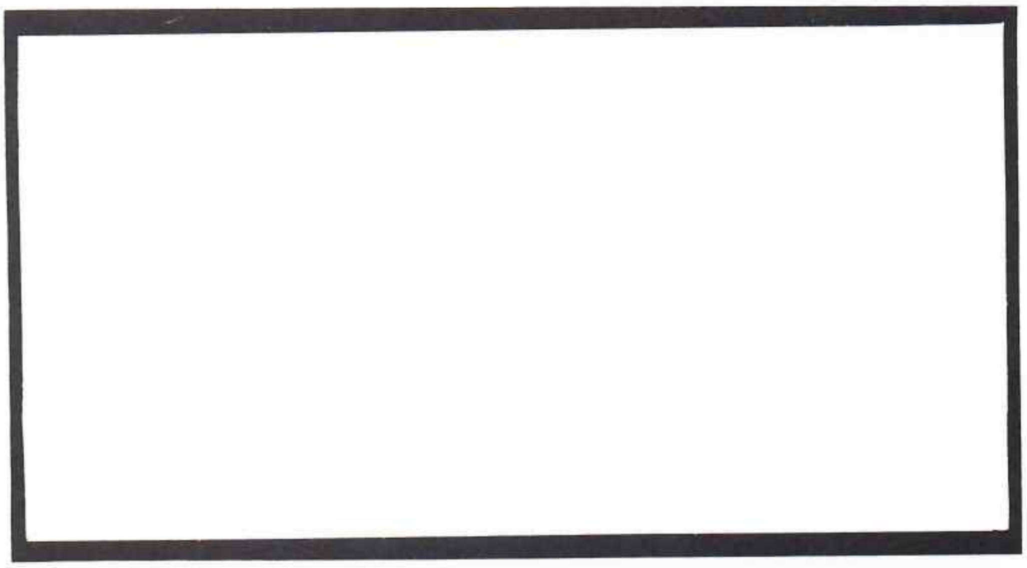


```
                                                                      FIRST
                          1. Create a customer file
                          2. Create a part description file
                          3. Create an employee file
                          4. Exit
                             Enter Option Number ... [2]
                                PART DESCRIPTION
 Part Number ... [    -    -  ]
 Description ... [                          ]
 Supplier ...... [                         ]
 Address ....... [                         ]
                 [                         ]
---------------------------------------------------------------------------
Salesperson [                        ]  Phone [   -   -    ]  Ext [    ]
Salesperson [                        ]  Phone [   -   -    ]  Ext [    ]
Salesperson [                        ]  Phone [   -   -    ]  Ext [    ]
Price ..... [$        .00]     per [       ]
                                                                      PARTS
                           1  Collect additional data
                           2  Return to menu
                           3  Exit
                           Enter Option Number ... [_]
                                                                       LAST
```

```
-------------------------------------------------------------------------------
! Named Data Information       !  Form name:          PARTS                   !
!                              !  Help form name:                             !
! Name      Data               !  First line:         7                       !
! NXTFRM    .NONE.             !  Last line:          18                      !
!                              !  Date created:       12-AUG-85               !
!                              !  Owner ID:           0                       !
!                              !  Form attributes:    None                    !
!                              !  Form length:        1102 bytes              !
!                              !  Number of fields:   19                      !
!                              !  Impure area size:   794 bytes               !
-------------------------------------------------------------------------------
```

Figure B-6. Part Description Form (PARTS) and Named Data for Part Description Form

You can also experiment with your own applications by using the Form Editor to create your own forms and add them to DEMLIB.FLB. Remember to modify the form named 'FIRST' so you can access your new form(s) from it.



```
-----------------------------------------------------------------------------
!                              !   Form name:          CLEARF              !
!                              !   Help form name:                         !
!                              !   First line:         1                   !
!                              !   Last line:          23                  !
!                              !   Date created:       23-SEP-85           !
!                              !   Owner ID:           0                   !
!                              !   Form attributes:    None                !
!                              !   Form length:        44 bytes            !
!                              !   Number of fields:   0                   !
!                              !   Impure area size:   220 bytes           !
-----------------------------------------------------------------------------
```

Figure B-7. Clear Form

Figure B-7 illustrates the Clear Form, which clears the screen for a new editing operation.

Appendix C
FMS Sample Application Programs

This appendix contains source listings of five FMS sample application programs. The FMS distribution kit contains the sources of these examples. They are built automatically as part of the FMS-11/RSX installation procedure. You can experiment with them while you are learning to use the FMS features.

Section C.1 contains listings of an FMS application that has been written in BASIC, COBOL, DIBOL, FORTRAN, and MACRO-11. All versions of the application use the same form library file, DEMLIB.FLB. This file is also in the FMS distribution kit.

The file names for the extended examples as stored on the FMS distribution kit are:

DEMLIB.FLB	The form library file for BASDEM, DBLDEM, FORDEM, COBDEM, and MACDEM.
CBLDEM.CBL	The COBOL version of the program.
BASDEM.B2S	The BASIC-PLUS-2 version of the program.
DBLDEM.DBL	The DIBOL-83 version of the program.
FORDEM.FTN	The FORTRAN IV and FORTRAN-77 version of the program.
MACDEM.MAC	The MACRO-11 version of the program.

For each version of the program, instructions about building a running application are included as comments at the beginning of the listing. Chapter 8, Building and Running FMS Application Systems, explains the building procedures in detail.

C.1 A Typical Application

This section lists BASIC-PLUS-2, COBOL-11, COBOL-81, DIBOL-83, FORTRAN IV, FORTRAN-77, and MACRO-11 versions of a simple user program that supports a multipurpose data-entry application.

The application supports different data-entry processes by using form descriptions that provide the details for each process. The program itself shows only the appropriate forms, depending on the process the operator chooses, and collects the operator's responses to each form. The application supports the following data-entry processes:

1. Create a customer file.
2. Create a part description file.
3. Create an employee file.
4. Exit.

C.2 Running the Programs

Before running any of the demo programs, you must have the form library DEMLIB.FLB in your system account. The demo programs are built as part of the FMS-11/RSX installation procedure.

Each task name assumes all libraries have been linked into the task, except for DIBOL-83, which has the DIBOL-83 resident library and the RMS resident library clustered.

C.2.1 Running the BASIC-PLUS-2 Version

```
RUN BASRMSDEM.TSK RET
```

C.2.2 Running the COBOL-11 Version

```
RUN C11RMSDEM.TSK RET
```

C.2.3 Running the COBOL-81 Version

```
RUN C81RMSDEM.TSK RET
```

C.2.4 Running the DIBOL-83 Version

```
RUN DBLRMSRES.TSK (RET)
```

C.2.5 Running the FORTRAN IV Version

```
RUN FORFCSDEM.TSK (RET)
```

C.2.6 Running the FORTRAN-77 Version

```
RUN F77RMSDEM.TSK (RET)
```

C.2.7 Running the MACRO-11 Version

```
RUN MACFCSDEM.TSK (RET)
```

C.3 Listing of the BASIC Program

```
00100   !*************************************************************
        !                                                            *
        !   BASDEM.B2S                                               *
        !                                                            *
        !                COPYRIGHT (C) 1985 BY                       *
        !   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.            *
        !                                                            *
00200   !*************************************************************
        !   BASDEM                                                   *
        !   BASIC DEMO PROGRAM                                       *
        !   AUTHOR:    CARRIE CODER                                  *
        !   DATE:      OCTOBER 24, 1985                              *
00500   !*************************************************************
        !
        !   Form record definition line
        !*************************************************************
        !
        MAP (POOL)              STRING  POOL                  =  256
        !*************************************************************
        !   Display Line for program debugging
        !*************************************************************
        !
        MAP (DISPLN)        STRING  DISPLAY_LINE          =  80
        MAP (DISPLN)        STRING  FILLER1               =  10,    &
                            STRING  MESS_LINE             =  30,    &
                            STRING  MESS_NBR              =  10,    &
                            STRING  FILLER2               =  30
        !
        !*************************************************************
        !   Codes for use by FMS-11 calls
        !*************************************************************
        !
        DECLARE   INTEGER   CONSTANT  EVENT_FLAG_TTY            =    5
        MAP       (FLDNM)   STRING    FIELD_NAME                =    6
        DECLARE   INTEGER             FIELD_TERMINATOR
        DECLARE   STRING    CONSTANT  FMS_LIBRARY      = "DEMLIB.FLB"
        DECLARE   INTEGER             FMS_STAT
        DECLARE   INTEGER   CONSTANT  FORM_LIB_CHANNEL          =    6
        DECLARE   INTEGER   CONSTANT  I_O_FUNCTION              =  768
        DECLARE   INTEGER   CONSTANT  FIRST_SIZE                =  500
        DECLARE   INTEGER   CONSTANT  LAST_SIZE                 =  400
        DECLARE   INTEGER   CONSTANT  UPDATE_SIZE               =  900
        DECLARE   BYTE                IMPURE_AREA_FIRST(500)
        DECLARE   BYTE                IMPURE_AREA_LAST(400)
        DECLARE   BYTE                IMPURE_AREA_UPDATE(900)
        DECLARE   INTEGER             RMS_STAT
        DECLARE   INTEGER   CONSTANT  TERMINAL_CHANNEL          =    5
        !
        !*************************************************************
        !  Temporary menu choices and current displayed form names
        !*************************************************************
        !
        MAP (DATFNM)    STRING   DATA_FILE_NAME             =   13
        MAP (FORMNM)    STRING   FORM_NAME                  =    6
        MAP (MENUNB)    STRING   MENU_CHOICE_NBR            =    1
        MAP (MENUTX)    STRING   MENU_CHOICE_TXT            =    6
        MAP (UPDFRM)    STRING   UPDATE_FORM                =    6
        !
        !*************************************************************
        !    Variables used to create a named data field name
        !*************************************************************
        !
        MAP (NDFLNM)    STRING   ND_FILE_NAME          =    5
        MAP (NDFLNM)    STRING   ND_FILE               =    4,  &
                        STRING   ND_FILE_NBR           =    1
        MAP (NDFILD)    STRING   ND_FIELD              =    6
        !
        !*************************************************************
        !   Initialize some of the variables
        !*************************************************************
        !
        ND_FILE      = "FILE"
        !
        !*************************************************************
        !   INITIALIZATION calls for FMS, etc.
        !*************************************************************
        !
        !   *   Attach the terminal, Enable Type-ahead          *
        !
03200   CALL WTQIO                     (I_O_FUNCTION,           &
                                        TERMINAL_CHANNEL,       &
                                        EVENT_FLAG_TTY)
        !
        !   *   Initialize FMS Impure Areas FIRST, LAST, UPDATE
        !   *       (in reverse order) to hold the forms
        !
03400   CALL FINIT                     (IMPURE_AREA_UPDATE(),   &
                                        UPDATE_SIZE,            &
                                        FMS_STAT )
        C=FN.26200_CHECK_STAT               ! Check value of FMS_STAT
        !
03600   CALL FINIT                     (IMPURE_AREA_LAST(),     &
                                        LAST_SIZE,              &
                                        FMS_STAT )
        C=FN.26200_CHECK_STAT               ! Check value of FMS_STAT
        !
```

```
03800   CALL FINIT                    (IMPURE_AREA_FIRST(),      &
                                       FIRST_SIZE,               &
                                       FMS_STAT )
        C=FN.26200_CHECK_STAT              ! Check value of FMS_STAT
        !
        !   *   Supply to the Form Driver the I/O Channel       *
        !   *     to use for reading the form library           *
        !
04000   CALL FLCHAN                   (FORM_LIB_CHANNEL)
        C=FN.26000_CHECK_STATUS_RESULT
        !
        !   *   Open the specified Form Library                 *
        !
04200   CALL FLOPEN                   (FMS_LIBRARY)
        C=FN.26000_CHECK_STATUS_RESULT
        !
        !   *   Clear the screen and display the 'FIRST' form   *
        !
        FORM_NAME = "FIRST"
04400   CALL FCLRSH                   (FORM_NAME)
        C=FN.26000_CHECK_STATUS_RESULT
        !
        !   *   Choose the Impure Area for the 'LAST' form
        !   *       and Display the 'LAST' menu on the screen
        !
04600   CALL FCHIMP                   (IMPURE_AREA_LAST() )
        C=FN.26000_CHECK_STATUS_RESULT
        !
        FORM_NAME = "LAST"
04800   CALL FSHOW                     (FORM_NAME)
        C=FN.26000_CHECK_STATUS_RESULT
        !
        ! *   Choose the Impure Area for the 'FIRST' form and
        ! *    allow the operator to select the data collection
        ! *    series.  Get the name of the first form to modify
        ! *    from named data.
        !
05000   CALL FCHIMP                   (IMPURE_AREA_FIRST() )
        C=FN.26000_CHECK_STATUS_RESULT
        !
        C=FN.22000_GET_MENU_OPTION
        UPDATE_FORM = MENU_CHOICE_TXT
06000   !**********************************************************
        !
        !
        WHILE MENU_CHOICE_TXT <> ".EXIT."
          !********************************************************
          !MAIN ROUTINE  -  Update until choice is 'EXIT' from menu
          !********************************************************
          !
          !*  Get the output file name from named data and open it
        !
        ND_FILE_NBR = MENU_CHOICE_NBR
10200   CALL FNDATA                     (ND_FILE_NAME,              &
                                            DATA_FILE_NAME)
          C=FN.26000_CHECK_STATUS_RESULT
          !
          !   *   Open the data file
          !
10400     OPEN DATA_FILE_NAME FOR OUTPUT AS FILE #1%,               &
              MAP             POOL
          !
          !   * Update this file until 'Collect Additional Data'
          !   *     is not the menu choice
          !
          MENU_CHOICE_NBR = "1"
          WHILE MENU_CHOICE_NBR = "1"
          !*********************************************************
          !   This is the data collection loop for one menu choice
          !*********************************************************
          !
          FORM_NAME = UPDATE_FORM
          !
          !   *   File may have more than one update form
          !   *     associated with it, update until next
          !   *     form or 'NXTFRM' = 'none'
          !
          WHILE FORM_NAME <> ".NONE."
                !************************************************
                !  Get one form full of data and write one record
                !************************************************
                !
                !   *   Clear the screen and show the form
                !
                CALL FCHIMP                     (IMPURE_AREA_UPDATE() )
                C=FN.26000_CHECK_STATUS_RESULT
12200           CALL FSHOW                      (FORM_NAME)
                C=FN.26000_CHECK_STATUS_RESULT
                !
                !   *   Get data for current form and output it
                !
                POOL = SPACE$(256)
12400           CALL FGETAL                     (POOL)
                C=FN.26000_CHECK_STATUS_RESULT
                PRINT #1%
                !
                !   *   Get name of next form.  If found, loop for
                !   *     more data, If '.NONE.' exit this loop
                !
                ND_FIELD = "NXTFRM"
12600           CALL FNDATA                     (ND_FIELD,          &
                                                 FORM_NAME)
                C=FN.26000_CHECK_STATUS_RESULT
                !
              NEXT
              !
              !    *    End of the form series.  Show 'LAST' form menu
              !    *       to determine if operator wants to continue
              !    *       with this data-type, return to main menu, or exit
              !
15000         CALL FCHIMP                    (IMPURE_AREA_LAST() )
              C=FN.26000_CHECK_STATUS_RESULT
              C=FN.22000_GET_MENU_OPTION
          NEXT
          !
          !    *    Close this output file
          !
          CLOSE #1%
          !
          !    *    Show the menu form for the operator to select the next
          !    *       file to update.  If named data from 'LAST' menu was
          !    *       '.EXIT.', don't prompt with 'FIRST' menu
          !
15200     IF MENU_CHOICE_TXT <> '.EXIT.'
          THEN
              CALL FCHIMP                (IMPURE_AREA_FIRST() )
              C=FN.26000_CHECK_STATUS_RESULT
              C=FN.22000_GET_MENU_OPTION
              UPDATE_FORM = MENU_CHOICE_TXT
          END IF
          NEXT
15400     !**************************************************************
          !    End this program routine
          !**************************************************************
          !
          GOTO S_30000_END_IT
          !**************************************************************
          !**************************************************************
22000     DEF FN.22000_GET_MENU_OPTION
          !**************************************************************
          !    GET A VALUE FROM THE MENU FORM
          !**************************************************************
          !
          !    *    Get a value from the form
          !
              C=FN.22400_GET_CHOICE
              WHILE FMS_STAT < 0
                  C=FN.22200_MENU_OPTION_ERROR
              NEXT
22008     FNEND
```

```
22200   DEF FN.22200_MENU_OPTION_ERROR
        !*************************************************************
        !
            DISPLAY_LINE = "Option not on list"
22202       CALL FPUTL                  (DISPLAY_LINE)
            C=FN.26000_CHECK_STATUS_RESULT
            C=FN.22400_GET_CHOICE
22208   FNEND
22400   DEF FN.22400_GET_CHOICE
        !*************************************************************
        !
        !   *   Get a value for menu 'FIRST' or 'LAST'
        !
            FIELD_NAME = "CHOICE"
22402       CALL FGET                   (MENU_CHOICE_NBR,           &
                                         FIELD_TERMINATOR,          &
                                         FIELD_NAME)
            C=FN.26000_CHECK_STATUS_RESULT
        !
        !  * Read named data, if number exists, menu choice was good
        !
22404       CALL FNDATA                 (MENU_CHOICE_NBR,           &
                                         MENU_CHOICE_TXT)
22406       CALL FSTAT                  (FMS_STAT,                  &
                                         RMS_STAT)
22408   FNEND
22409   !*************************************************************
26000   DEF FN.26000_CHECK_STATUS_RESULT
        !*************************************************************
        !   this section is for checking the status of the FMS calls
        !*************************************************************
            !
            !   * Check to see if there was an FMS or RMS error     *
            !
26002       CALL FSTAT                  (FMS_STAT,                  &
                                         RMS_STAT)
            !
26006       C=FN.26200_CHECK_STAT
26008   FNEND


26200   DEF FN.26200_CHECK_STAT
        !*************************************************************
        !
        !   *   Some calls return status in the call.  For those
        !   *       calls the 'FSTAT' call is not necessary
        !
        !
26206       IF  FMS_STAT <  0 OR RMS_STAT <  0  THEN
                C=FN.26220_FMS_ERROR
            !
26208   FNEND
```

```
26220   DEF FN.26220_FMS_ERROR
             !
             ! * Display an FMS error number and stop the program *
             !
26222        CALL FLCLOS()
             PRINT " "
             PRINT " FMS STATUS....", FMS_STAT
             PRINT " RMS STATUS....", RMS_STAT
             STOP
26228   FNEND
26229   !************************************************************


30000   S_30000_END_IT:
        !************************************************************
        !    End this program routine
        !************************************************************
        !
             FORM_NAME = "CLEARF"
             CALL FCLRSH                      (FORM_NAME)
             CALL FLCLOS()
             !
32767   END
```

C.4 Listing of the COBOL Program

```
*    COBDEM.CBL
*
*              COPYRIGHT (C) 1985 BY
*    DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
 IDENTIFICATION DIVISION.
*
*    Program was written in conventional (ANSI standard) format
*
*    To compile this program and create COBOL-11 object code:
*
*    > CBL C11DEM,C11DEM=COBDEM/CVF
*
*    To compile this program and create COBOL-81 object code:
*
*    > C81 C81DEM,C81DEM=COBDEM/CVF
*
 PROGRAM-ID.     COBDEM.
 AUTHOR.         CARRIE CODER.
 DATE-WRITTEN.   24-OCT-1985.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
*
 SELECT OUTPUT-FILE ASSIGN TO "SY:".
*
 DATA DIVISION.
 FILE SECTION.
*
*    Create a sequential file for output of form data.
*
 FD  OUTPUT-FILE
     LABEL RECORDS ARE STANDARD
     VALUE OF ID IS DATA-FILE-NAME.
 01  POOL                          PIC X(256).
*
 WORKING-STORAGE SECTION.
*
*    Temporary variable for display
*
 01  DISPLAY-LINE.
     10  FILLER                            PIC X(2)
             VALUE SPACES.
     10  MESS-LINE                         PIC X(30).
     10  MESS-NBR                          PIC -9(5).
*
*    Set up variables to be used in FMS Form Driver Calls.
```

```
*
 01  EVENT-FLAG-TTY                        PIC  9(2)   COMP
          VALUE 1.
 01  FIELD-NAME                            PIC X(6).
 01  FIELD-TERMINATOR                      PIC S9(4)  COMP.
 01  FMS-LIBRARY                           PIC X(10)
         VALUE "DEMLIB.FLB".
 01  FMS-STATUS-BLOCK.
     10  FMS-STATUS                        PIC S9(4)  COMP.
     10  RMS-STATUS                        PIC S9(4)  COMP.
 01  FORM-LIB-CHANNEL                      PIC  9(2)  COMP
          VALUE   3.
 01  I-O-FUNCTION                          PIC  9(3)  COMP
          VALUE 768.
 01  IMPURE-AREA-FIRST                     PIC X(1000).
 01  IMPURE-AREA-LAST                      PIC X(1000).
 01  IMPURE-AREA-UPDATE                    PIC X(1000).
 01  IMPURE-AREA-SIZE                      PIC  9(4)  COMP
          VALUE 1000.
 01  TERMINAL-CHANNEL                      PIC  9(2)  COMP
          VALUE   1.
*
 01  FMS-ERROR-LINE.
     10  FILLER                            PIC X(2)
         VALUE SPACES.
     10  FILLER                            PIC X(15)
         VALUE "FMS STATUS.....".
     10  FMS-STAT-OUT                      PIC -9(9).
     10  FILLER                            PIC X(2)
         VALUE SPACES.
     10  FILLER                            PIC X(15)
        VALUE "RMS STATUS.....".
     10  RMS-STAT-OUT                      PIC -9(9).
*
*    Temporary menu choices and current displayed form names
*
 01  DATA-FILE-NAME                        PIC X(13).
 01  FORM-NAME                             PIC X(6).
 01  MENU-CHOICE-NBR                       PIC X(1).
 01  MENU-CHOICE-TXT                       PIC X(6).
 01  UPDATE-FORM                           PIC X(6).
*
*    Variables used to create a named data field name.
*
 01  ND-FILE-NAME.
     10  FILLER                            PIC X(4)
         VALUE "FILE".
     10  ND-FILE-NBR                       PIC X(1).
     10  FILLER                            PIC X(1)
         VALUE SPACE.
 01  ND-FIELD                              PIC X(6).
*****************************************************************
```

```
*
*
 PROCEDURE DIVISION.
*
*
 A000-MAIN-CONTROL.
     PERFORM I000-INITIALIZE   THRU I000-EXIT.
     PERFORM M000-MAIN-ROUTINE THRU M000-EXIT
        UNTIL MENU-CHOICE-TXT = ".EXIT.".
     PERFORM Z000-END-IT       THRU Z000-EXIT.
 A000-EXIT.
***************************************************************
*
*
 I000-INITIALIZE.
*
* Attach the terminal, enable type-ahead
*
 CALL "WTQIO"    USING  BY REFERENCE  I-O-FUNCTION,
                        BY REFERENCE  TERMINAL-CHANNEL,
                        BY REFERENCE  EVENT-FLAG-TTY.
*
* Initialize FMS Impure Area (in reverse order) to hold the forms
*
 CALL "FINIT"    USING  BY DESCRIPTOR IMPURE-AREA-UPDATE,
                        BY REFERENCE  IMPURE-AREA-SIZE,
                        BY REFERENCE  FMS-STATUS-BLOCK.
 PERFORM S620-CHECK-STAT          THRU S620-EXIT.
*
 CALL "FINIT"    USING  BY DESCRIPTOR IMPURE-AREA-LAST,
                        BY REFERENCE  IMPURE-AREA-SIZE,
                        BY REFERENCE  FMS-STATUS-BLOCK.
 PERFORM S620-CHECK-STAT          THRU S620-EXIT.
*
 CALL "FINIT"    USING  BY DESCRIPTOR IMPURE-AREA-FIRST,
                        BY REFERENCE  IMPURE-AREA-SIZE,
                        BY REFERENCE  FMS-STATUS-BLOCK.
 PERFORM S620-CHECK-STAT          THRU S620-EXIT.
*
*    Supply to the Form Driver the I/O Channel
*        to use for reading the form library
*
 CALL "FLCHAN"   USING  BY REFERENCE  FORM-LIB-CHANNEL.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
*    Open the specified form library
```

```
*
 CALL "FLOPEN"   USING  BY DESCRIPTOR FMS-LIBRARY.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
*    Display the 'FIRST' menu on the screen
*    Since 'FIRST' impure area was attached last, should still be current
*
 MOVE "FIRST " TO FORM-NAME.
 CALL "FCLRSH"   USING  BY DESCRIPTOR FORM-NAME.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
*    Choose the Impure Area for the 'LAST' form
*    and Display the 'LAST' menu on the screen
*
 CALL "FCHIMP"   USING  BY DESCRIPTOR IMPURE-AREA-LAST.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
 MOVE "LAST  " TO FORM-NAME.
 CALL "FSHOW"   USING  BY DESCRIPTOR FORM-NAME.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
*    Choose the Impure Area for the 'FIRST' form and
*    Allow the operator to select the data collection series.
*    Get the name of the first form to modify from named data.
*
 CALL "FCHIMP"    USING  BY DESCRIPTOR IMPURE-AREA-FIRST.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
 PERFORM S200-GET-MENU-OPTION     THRU S200-EXIT.
 MOVE MENU-CHOICE-TXT TO UPDATE-FORM.
*
 I000-EXIT.
**********************************************************************
*
*
 M000-MAIN-ROUTINE.
*
*    Get the output file name from named data and open it.
*
 MOVE MENU-CHOICE-NBR TO ND-FILE-NBR.
 MOVE SPACES TO DATA-FILE-NAME.
 CALL "FNDATA"   USING BY DESCRIPTOR ND-FILE-NAME,
                       BY DESCRIPTOR DATA-FILE-NAME.
 PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
 OPEN OUTPUT OUTPUT-FILE.
*
*    Update this file until 'Collect Additional Data' is not the choice.
*
 MOVE "1" TO MENU-CHOICE-NBR.
 PERFORM M200-UPDATE-ONE-FILE THRU M200-EXIT
      UNTIL MENU-CHOICE-NBR NOT = "1".
```

```
*
*    Close this output file.
*
   CLOSE OUTPUT-FILE.
*
*    Show the menu form for operator to select the next file
*        to update.  If named data from 'LAST' menu was ".EXIT.",
*        don't prompt with 'FIRST' menu
*
  IF MENU-CHOICE-TXT NOT = ".EXIT."
      CALL "FCHIMP"    USING  BY DESCRIPTOR IMPURE-AREA-FIRST
      PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT
     PERFORM S200-GET-MENU-OPTION THRU S200-EXIT
      MOVE MENU-CHOICE-TXT TO UPDATE-FORM.
M000-EXIT.
***********************************************************************
*
*
 M200-UPDATE-ONE-FILE.
*
*    This is the data collection loop.
*
     MOVE UPDATE-FORM TO FORM-NAME.
*
*    File may have more than one form, update until next form = 'none'
*
  PERFORM M220-COLLECT-DATA THRU M220-EXIT
      UNTIL FORM-NAME = ".NONE.".
*
*    End of the form series.  Show 'LAST' form menu to determine
*        if operator wants to continue with this data-type, return
*        to main menu, or exit.
*
  CALL "FCHIMP"    USING  BY DESCRIPTOR IMPURE-AREA-LAST.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
  PERFORM S200-GET-MENU-OPTION THRU S200-EXIT.
*
 M200-EXIT.
***********************************************************************
*
*
 M220-COLLECT-DATA.
*
*    Clear the screen and show the form.
*
  CALL "FCHIMP"  USING BY DESCRIPTOR IMPURE-AREA-UPDATE.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
  CALL "FSHOW"   USING BY DESCRIPTOR FORM-NAME.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
```

```
*    Get data for current form and output it.
*
  MOVE SPACES TO POOL.
  CALL "FGETAL"   USING BY DESCRIPTOR POOL.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
  WRITE POOL.
*
*    Get name of next form.  If found, loop for more data.
*
  MOVE "NXTFRM" TO ND-FIELD.
  CALL "FNDATA"   USING BY DESCRIPTOR ND-FIELD,
                        BY DESCRIPTOR FORM-NAME.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
 M220-EXIT.
************************************************************************
*
*
 S200-GET-MENU-OPTION.
*
*    Get a value from the form
*
  PERFORM S240-GET-CHOICE  THRU S240-EXIT.
  IF FMS-STATUS < 0
      PERFORM S220-MENU-OPTION-ERROR THRU S220-EXIT
          UNTIL FMS-STATUS > 0.
 S200-EXIT.
*
 S220-MENU-OPTION-ERROR.
  MOVE "Option not on list" TO DISPLAY-LINE.
  CALL "FPUTL"    USING BY  DESCRIPTOR DISPLAY-LINE.
  PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
  PERFORM S240-GET-CHOICE          THRU S240-EXIT.
 S220-EXIT.
*
 S240-GET-CHOICE.
  MOVE "CHOICE" TO FIELD-NAME.
  CALL "FGET"     USING  BY  DESCRIPTOR MENU-CHOICE-NBR,
                         BY  REFERENCE  FIELD-TERMINATOR,
                         BY  DESCRIPTOR FIELD-NAME.
     PERFORM S600-CHECK-STATUS-RESULT THRU S600-EXIT.
*
  CALL "FNDATA"   USING  BY  DESCRIPTOR MENU-CHOICE-NBR,
                         BY  DESCRIPTOR MENU-CHOICE-TXT.
  CALL "FSTAT"    USING  BY  REFERENCE FMS-STATUS
                         BY  REFERENCE RMS-STATUS.
 S240-EXIT.
*******************************************************************
```

```
*
*
 S600-CHECK-STATUS-RESULT.
*-----------------------------------------------------------------
*  This section is for checking the status of the FMS calls
*-----------------------------------------------------------------
*
*   check to see if there was an FMS or RMS error
*
  CALL "FSTAT"    USING  BY REFERENCE FMS-STATUS
                         BY REFERENCE RMS-STATUS.
  PERFORM S620-CHECK-STAT THRU S620-EXIT.
 S600-EXIT.
*
 S620-CHECK-STAT.
*
*    Some calls return status in the call.  For those calls,
*        the 'FSTAT' call is not necessary.
*
  IF      FMS-STATUS LESS THAN ZERO  OR
          RMS-STATUS LESS THAN ZERO
          PERFORM S622-FMS-ERROR THRU S622-EXIT.
 S620-EXIT.
*****************************************************************
*
*
 S622-FMS-ERROR.
*
*    Display the FMS error number and stop the program
*
  MOVE FMS-STATUS TO FMS-STAT-OUT.
  MOVE RMS-STATUS TO RMS-STAT-OUT.
  CALL "FPUTL"    USING  BY DESCRIPTOR FMS-ERROR-LINE.
  CALL "FLCLOS".
  STOP RUN.
 S622-EXIT.
*
*
************************************************************
*
*    Close form library and exit.
*
 Z000-END-IT.
  MOVE "CLEARF"  TO  FORM-NAME.
  CALL "FCLRSH"   USING  BY DESCRIPTOR FORM-NAME.
  CALL "FLCLOS".
  STOP RUN.
 Z000-EXIT.
*
************************************************************
```

C.5 Listing of the DIBOL Program

```
START                                   ; FMS Demonstration Program


;
; DBLDEM.DBL
;
;
;                          COPYRIGHT (C) 1985 BY
;              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
;
;
; MODULE:          DBLDEM
;
; VERSION:         V2.0
;
; AUTHOR:          Megan
;
; DATE:            01-September-85
;
;
; This is the DIBOL demonstration program for FMS illustrating
; a simple form-driven, data-entry application.
;
;
; The following requirements are unique to DIBOL programs using FMS:
;
;        The name of the form library opened by FLOPEN must have a
;        trailing space following the name.
;
;

RECORD   IMPURE                         ; Form Driver impure area.
,        A875

RECORD   IFIRST                         ; Impure area for FIRST form
,        A475

RECORD   ILAST                          ; Impure area for LAST form
,        A350                           ; RECORD IMPURE
                                        ; Form Driver impure area.
                                        ; Form Driver Terminal Control Block.

RECORD
FCHAN,   D1,     1                      ; Channel for form library.
DCHAN,   D1,     2                      ; Channel for data file.
CTR,     D4                             ; General counter
NAMED,   A7                             ; Named_data name
DATA,    A256                           ; Area where data from form is stored
CHOICE,  A6                             ; User choice goes here.
```

```
TERM,     D3                        ; Terminator.
FORM,     A16                       ; Initial form name.
FILE,     A16                       ; Output file name.
FORM1,    A16                       ; Current form name.

RECORD    STATUS
FDVSTS,   D4,    0
RMSSTS,   D4,    0

RECORD    STAMSG
HD,       A15,  'FSTAT RETURNED '
ER,       D3
CL,       A6

RECORD    FTLMSG
FHD,      A21,  'FATAL ERROR RETURNED '
FER,      D3
FCL,      A6

          PROC

;
; Initialize Form Driver and open form library.
;
          XCALL FINIT (IMPURE,875,FDVSTS) ; Define forms impure area.
          CALL  L2001
          XCALL FINIT (ILAST, 350,FDVSTS) ; Define forms impure area.
          CALL  L2001
          XCALL FINIT (IFIRST,475,FDVSTS) ; Define forms impure area.
          CALL  L2001

          XCALL FATTCH()                  ; Attach terminal.
          CALL  L2000
          XCALL FLCHAN(FCHAN)             ; Define I/O channel for library.
          CALL  L2000
          XCALL FLOPEN('DEMLIB ')         ; Open form library (trailing
                                          ; space necessary).
          CALL  L2000
;
; Display menu form for operator to select data collection series.
; Get the first form name from named data.
;
          XCALL FCLRSH('FIRST')           ; Get the menu form.
          CALL  L2000                     ; Check the status.
          XCALL FCHIMP(ILAST)             ; Get correct impure area
          CALL  L2000                     ; Check the status.
          XCALL FSHOW('LAST')             ; Get the menu form.
          CALL  L2000                     ; Check the status.
L500,     XCALL FCHIMP(IFIRST)            ; Get correct impure area
          CALL  L2000                     ; Check the status.
```

```
L510,    XCALL FGET(CHOICE,TERM,'CHOICE'); Get the user's choice.
         CALL  L2000                     ; Check the status.
         XCALL FNDATA(CHOICE,FORM        ; Get named data for valid
                                         ; responses.
         XCALL FSTAT(FDVSTS)             ; Get the status.
         IF    (FDVSTS.GT.0) GOTO L570   ; Br if no error.
         XCALL FPUTL('Illegal Choice')   ; Else output error message
         GOTO  L510                      ; and get choice again.
L570,    IF (FORM .EQ. '.EXIT.') GOTO L1090 ; If form name is
                                            ; '.EXIT.',
                                         ; terminal operation is done.
;
; Get data file name from named data and open it.
;

         CTR   =1                        ; Initialize cell counter.
LP,      NAMED(CTR,CTR)=CHOICE(CTR,CTR)  ; Synchronize.
         INCR CTR                        ; Increment counter.
         IF (CHOICE(CTR,CTR).NE.' ') GOTO LP  ; Continue if not end of string.
         NAMED(CTR,CTR)='F'              ; Else add the 'F'.

         XCALL FNDATA(NAMED,FILE)        ; Get file name from named data.
         OPEN  (DCHAN,O,FILE)            ; Open the output file.

;
; This is the data collection loop.
;

L680,    FORM1 =FORM                     ; Set current form = first in series.
         XCALL FCHIMP(IMPURE)            ; Impure area for data forms
         CALL  L2000
L720,
         XCALL FSHOW(FORM1)              ; Display the form.
         CALL  L2000                     ; Check the status.

         XCALL FGETAL(DATA)              ; Get & output data for current form.
         CALL  L2000                     ; Check status.
         WRITES (DCHAN,DATA)             ; Write to output file.

         XCALL FNDATA('NXTFRM',FORM1)    ; Get name of next form.
         IF (FORM1.NE.'.NONE.') GOTO L720; If found, loop for more
                                         ; data.

;
; End of the form series. Display menu to determine if user is done.
;
         XCALL FCHIMP(ILAST)
         CALL  L2000                     ; Get the status.
```

```
L880,     XCALL FGET(CHOICE,TERM,'CHOICE') ; Get the next choice.
          CALL  L2000                      ; Get the status.

          IF    (CHOICE.EQ.'1') GOTO L680  ; If response = "1" repeat

                                           ;   data collection loop.

;
; Get named data corresponding to response.
; Get field again if illegal response.
; Close output file for valid response other than repeat.
;

          XCALL FNDATA(CHOICE,FORM1)       ; Get named data.
          XCALL FSTAT(FDVSTS)              ; Get the status.
          IF (FDVSTS.GT.0) GOTO L1000      ; Br if no error.
          XCALL FPUTL('Illegal Choice')    ; Else output error message.
          GOTO L880                        ; Get another choice.
L1000,    CLOSE (DCHAN)                    ; Close the output channel.

;
; If named data is '.EXIT.', terminal operator is done.
; Otherwise, display menu form again.
;

          IF (FORM1.NE.'.EXIT.') GOTO L500

L1090,    XCALL FCLRSH('CLEARF')           ; Clear screen on exit.
          XCALL FLCLOS                     ; Close form library.
          XCALL FDETCH                     ; Detach terminal.
          STOP                             ; Exit.

;
; Output message and exit if I/O error returned from Form Driver.
; This is the only error expected in a debugged application.
;

L2000,    XCALL FSTAT(FDVSTS,RMSSTS)       ; Get the status.
L2001,    IF (FDVSTS.LT.0) GOTO EXIT       ; Return if no error.
          ER=FDVSTS                        ; COPY STATUS
;         XCALL FPUTL(STAMSG)
          RETURN

EXIT,     FER=FDVSTS
          XCALL FPUTL(FTLMSG)              ; Else display the error
                                           ; message.
          XCALL FDETCH                     ; Detach terminal.
          XCALL FLCLOS                     ; Close library channel
          STOP                             ; Exit.
          END
```

C.6 Listing of the FORTRAN Program

```
C       !*********************************************************
C       !                                                        *
C       !   FORDEM.FTN                                           *
C       !                                                        *
C       !                 COPYRIGHT (C) 1985 BY                  *
C       !   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.        *
C       !                                                        *
C       !*********************************************************
C       !   FORDEM                                               *
C       !   FORTRAN DEMO PROGRAM                                 *
C       !   AUTHOR:    CARRIE CODER                              *
C       !   DATE:      OCTOBER 24, 1985                          *
C       !*********************************************************
C       !
C       !   Form record definition line
C       !*********************************************************
C       !
         BYTE                     POOL(256)
C       !
C       !*********************************************************
C       !   Display Line for program debugging
C       !*********************************************************
C       !
         BYTE                     DISPLN(80)
C       !
C       !*********************************************************
C       !   Codes for use by FMS-11 calls
C       !*********************************************************
C       !
         INTEGER                  chan
         INTEGER                  ioflag
         INTEGER                  iofunc
         BYTE                     fid(7)
         BYTE                     flnm(11)
         BYTE                     fname(7)
         BYTE                     impurf(500)
         BYTE                     impurl(400)
         BYTE                     impuru(900)
         INTEGER                  isizef
         INTEGER                  isizel
         INTEGER                  isizeu
         INTEGER                  status
         INTEGER                  stat2
         INTEGER                  term
         INTEGER                  ttchan
C       !
C       !
C       !*********************************************************
C       !Temporary menu choices and current displayed form names
C       !*********************************************************
```

```
C       !
         BYTE                     DFNAME(14)
         BYTE                     MENUNB
         BYTE                     MENUTX(7)
         BYTE                     UPDFRM(7)
C       !
C       !*****************************************************
C       !    Variables used to create a named data field name
C       !*****************************************************
C       !
         BYTE                     NDNAME(6)
         BYTE                     NDFLD(7)
C       !
C       !*****************************************************
C       !   Initialize some of the variables
C       !*****************************************************
C       !
         DATA      ioflag     /     5 /
         DATA      chan       /     6 /
         DATA      iofunc     /   768 /
         DATA      isizef     /   500 /
         DATA      isizel     /   400 /
         DATA      isizeu     /   900 /
         DATA      ttchan     /     5 /
C
         CALL      SCOPY      ( 'DEMLIB.FLB', flnm,    10)
         CALL      SCOPY      ( 'FILE ',      NDNAME,  5)
C       !
C       !*****************************************************
C       !   INITIALIZATION calls for FMS, etc.
C       !*****************************************************
C       !
C       !   *   Attach the terminal, Enable Type-ahead         *
C       !
         CALL WTQIO            (iofunc,  ttchan,   ioflag)
C       !
C       !   *   Initialize FMS Impure Areas (FIRST, LAST, UPDATE)
C       !   *     (in reverse order) to hold the forms
C       !
         CALL FINIT           ( impuru,  isizeu,  status )
         CALL CHFMST          ( status,  stat2 )
C       !
         CALL FINIT           ( impurl,  isizel,  status )
         CALL CHFMST          ( status,  stat2 )
C       !
         CALL FINIT           ( impurf,  isizef,  status )
         CALL CHFMST          ( status,  stat2 )
C       !
C       !   *   Supply to the Form Driver the I/O Channel       *
C       !   *     to use for reading the form library           *
C       !
         CALL FLCHAN          ( chan )
         CALL STATCK          ( status,   stat2 )
```

```
C        !
C        !   *    Open the specified Form Library                        *
C        !
          CALL FLOPEN         ( flnm )
          CALL STATCK         ( status,   stat2 )
C        !
C        !   *    Clear the screen and display the 'FIRST' form     *
C        !
          CALL SCOPY          ( 'FIRST ', fname,   6 )
          CALL FCLRSH         ( fname )
          CALL STATCK         ( status,   stat2 )
C        !
C        !   *    Choose the Impure Area for the 'LAST' form
C        !   *         and Display the 'LAST' menu on the screen
C        !
          CALL FCHIMP         ( impurl )
          CALL STATCK         ( status,   stat2 )
C        !
          CALL SCOPY          ( 'LAST  ', fname,   6)
          CALL FSHOW          ( fname )
          CALL STATCK         ( status,   stat2 )
C        !
C        !   *    Choose the Impure Area for the 'FIRST' form and
C        !   *    allow the operator to select the data collection
C        !   *    series.  Get the name of the first form to modify
C        !   *    from named data.
C        !
300       CALL FCHIMP         ( impurf )
          CALL STATCK         ( status,   stat2 )
C        !
C        !   *    Get first menu choice, if '.EXIT.' skip to end
C        !
          CALL MENUCH         ( MENUNB,   MENUTX )
          IF (SCOMP (MENUTX, '.EXIT.') .EQ. 0) GO TO 900
C        !
          CALL SCOPY          ( MENUTX,   UPDFRM,   6)
C        !
C        !   *    *    End of Initialization Calls     *    *
C        !
C        !***********************************************************


C        !***********************************************************
C        !   MAIN ROUTINE LOOP
C        !***********************************************************
C        !
C        !   ***************************************************** 400
C        !   START - Loop until menu choice is '.EXIT.'
C        !   ***************************************************** 400
C        !
C        !   *    Get the output file name from named data and open it
C        !
  400    NDNAME(5) = MENUNB
```

```
          CALL FNDATA       ( NDNAME,   DFNAME )
          CALL STATCK       ( status,   stat2 )
C         !
C         !   *   Open the data file
C         !
          OPEN  ( UNIT=1,       NAME=DFNAME,  INITIALSIZE=10,
     2            TYPE='NEW',   RECORDSIZE=128 )
C         !
C         ! * Initialize menu choice to '1' or 'Collect Additional Data'
C         !
          MENUNB = '1'
C         !
C         !   **************************************************** 500
C         !   START - Loop until menu choice is not
          !             '1' - 'Additional Data'
C         !   **************************************************** 500
C         !
  500         CALL SCOPY     ( UPDFRM,   fname,   6)
C         !
C         !   **************************************************** 600
C         !   START - File may have more than one update form
          !   associated with it, Loop until named data
C         !   has '.NONE.' as NXTFRM
C         !   **************************************************** 600
C         !
C         !   *   Clear the screen and show UPDFRM form or NXTFRM form
C         !
  600              CALL FCHIMP          ( impuru )
                   CALL STATCK          ( status,    stat2 )
                   CALL FSHOW           ( fname )
                   CALL STATCK          ( status,    stat2 )
C                 !
C                 !   *   Get data for current form and output it
C                 !
                   DO 605 I=1,128
                   POOL(I) = 0
  605              CONTINUE
C
                   CALL FGETAL          ( POOL )
                   CALL STATCK          ( status,    stat2 )
  615              FORMAT (128A1)
                   WRITE (1,615) (POOL(I), I=1,128)
C              !
C              !   *   Get name of next form.  If found, loop for more data
C              !   *                           If '.NONE.' exit this loop
C              !
                   CALL SCOPY           ( 'NXTFRM', NDFLD,    6 )
                   CALL FNDATA          ( NDFLD,    fname )
                   CALL STATCK          ( status,   stat2 )
C                  !
               IF (SCOMP (fname, '.NONE.') .NE. 0) GO TO 600
C         !   ********************************************************** 600
C         !   END - Loop until there are no more forms for this data file
```

```
C       !   ************************************************************ 600
C       !
C       !   *   End of the form series.  Show 'LAST' form menu to determine
C       !   *     if operator wants to continue with this data-type, return
C       !   *     to main menu, or exit
C       !
  700          CALL FCHIMP     ( impurl )
               CALL STATCK     ( status,   stat2 )
               CALL MENUCH     ( MENUNB,   MENUTX )
C       !
          IF (MENUNB .EQ. '1') GO TO 500
C   !   ****************************************************************** 500
C   !   END - Loop until menu choice is not '1' - 'Additional Data'
C   !   ****************************************************************** 500
C   !
C   !   *   Close this output file
C   !
  800     CLOSE (UNIT=1)
C       !
C       !   *   Show the menu form for the operator to select the next
C       !   *     file to update.  If named data from 'LAST' menu was
C       !   *     '.EXIT.', don't prompt with 'FIRST' menu
C       !
  900     IF (SCOMP (MENUTX, '.EXIT.') .NE. 0) GO TO 300
C       !   ************************************************************ 300
C       !   END - Loop until menu choice is '.EXIT.'
C       !   ************************************************************ 300
C
C
C       !******************************************************************
C       !   End this program routine
C       !******************************************************************
C       !
C       !   *   Clear screen, show empty form, close form library
C       !
          CALL SCOPY          ( 'CLEARF', fname,   6 )
          CALL FCLRSH         ( fname )
          CALL FCLOS
C       !
          END


C       !******************************************************************
          SUBROUTINE MENUCH              ( MENUNB,   MENUTX )
C       !   GET A VALUE FROM THE MENU FORM
C       !******************************************************************
C       !
          BYTE                 MENUNB
          BYTE                 MENUTX(7)
          BYTE                 DISPLN(80)
          BYTE                 fid(7)
          INTEGER              status
          INTEGER              stat2
```

```
C       !
C       !   *   Get a value for menu 'FIRST' or 'LAST'
C       !
         CALL SCOPY         ( 'CHOICE', fid,   6 )
  500    CALL FGET          ( MENUNB,   term,   fid)
         CALL STATCK        ( status,   stat2 )
C       !
C       !   *   Read named data, if number exists, menu choice was good
C       !
         CALL FNDATA        ( MENUNB,   MENUTX )
         CALL FSTAT         ( status,   stat2 )
            IF (status .GT. 0 ) GO TO 550
C       !
C       !   *   There was an error
C       !
         CALL SCOPY ('Option not on list', DISPLN, 18)
         CALL FPUTL         ( DISPLN )
         CALL STATCK        ( status,   stat2 )
C       !
        GO TO 500
C       !
  550   RETURN
        END


C       !*************************************************************
        SUBROUTINE STATCK ( status, stat2 )
C       !   this subroutine will check the status of the FMS calls
C       !*************************************************************
C       !
         INTEGER                status
         INTEGER                stat2
C       !
C       !   *   Check to see if there was an FMS or RMS error        *
C       !
         CALL FSTAT         ( status,   stat2 )
C       !
         CALL CHFMST        ( status,   stat2 )
         RETURN
         END


C       !*************************************************************
        SUBROUTINE CHFMST               ( status,   stat2 )
C       !   *  Some calls return status in the call.  For those
C       !   *       calls the 'FSTAT' call is not necessary
C       !*************************************************************
C       !
         INTEGER                status
         INTEGER                stat2
C       !
            IF  (status .GT. 0 ) GO TO 650
```

```
C       !
C       !   *   Display an FMS error number and stop the program *
C       !
        CALL FLCLOS
  645   FORMAT (/, 2X, 'FMS STATUS...', I5,
     2          /, 2X, 'RMS STATUS...', I5)
        TYPE 645, status, stat2
        STOP
C       !
  650   RETURN
        END


C       !*****************************************************
        SUBROUTINE SCOPY   ( SRC,       DST,       LEN)
C       !   *   Copy a string of a specified length
C       !*****************************************************
C       !
C       !   *   SRC = source byte string
C       !   *   DST = destination byte string to be ended by a zero
C       !   *   LEN = number of characters to copy
C       !
        BYTE SRC(1), DST(1)
        INTEGER LEN
C
C       !   *   Copy source to destination for length
C
        DO 10 I = 1, LEN
        DST(I) = SRC(I)
   10   CONTINUE
C
C       !   *   End destination string with zero byte
C
        DST(LEN+1) = 0
        RETURN
        END


C       !*****************************************************
        INTEGER FUNCTION SCOMP       ( SRC1,     SRC2)
C       !   *   Compare two strings
C       !*****************************************************
C
C   !   *    SRC1 = first comparand byte string ended by a zero
C   !   *    SRC2 = second comparand byte string ended by a zero
C
C   !   * Value of function is zero for equal, nonzero for not equal
C   !   * Compare returns failure if string lengths are not the same
C
        BYTE SRC1(1), SRC2(1)
C
C   ! * Compare until either string ends in zero byte or does not match
```

```
C
        I = 1
   10   IF (SRC1(I) .EQ. 0 .AND. SRC2(I) .EQ. 0) GOTO 20
        IF (SRC1(I) .NE. SRC2(I)) GOTO 30
        I = I + 1
        GOTO 10
C
C      !   *    Return success
C
   20   SCOMP = 0
        RETURN
C
C      !   *    Return failure
C
   30   SCOMP = 1
        RETURN
C
        END
```

C.7 Listing of the MACRO-11 Program

```
$ TY MACDEM.MAC
        .TITLE MACDEM - FMS DEMONSTRATION SUBROUTINE
;       .LIST MEB
;
; MACDEM.MAC
;
;
;                           COPYRIGHT (C) 1985 BY
;              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
;
;
; MODULE:       MACDEM
;
; VERSION:      V02.00
;
; AUTHOR:       MEGAN
;
; DATE:         17-SEPTEMBER-85
;
;

        .ENABL  LC                      ; Allow lower case source text


        .MCALL  $FDV,$FDVDF             ; Identify Form Driver macro calls
        .MCALL  QIOW$S,EXIT$S           ; RSX I/O related macros
        .MCALL  FINIT$,FSRSZ$           ; FCS macros
        .MCALL  FDBDF$,FDRC$A,FDOP$A,NMBLK$
        .MCALL  FDAT$A,OPEN$W,PUT$,CLOSE$
        FSRSZ$  1                       ; Set the file storage region

        $FDVDF                          ; Init the Form Driver definitions
;
; Equated symbols
;
        .PSECT  MACDAT,D,REL

        ISIZ=1024.                      ; Size of FDV impure area
        IN$CHN=1                        ; Input channel number (Form Library)
        OU$CHN=2                        ; Output channel number (Output File)

        .SBTTL Local data
EXTNAM: .ASCII  /.EXIT./                ; Exit name
NONNAM: .ASCII  /.NONE./                ; No more forms in series
FSTNAM: .ASCII  /FIRST /                ; ASCII form name
LSTNAM: .ASCII  /LAST  /                ; ASCII form name
CLRSCR: .ASCII  /CLEARF/                ; ASCII form name
CHCNAM: .ASCII  /CHOICE/                ; ASCII field name
```

```
NXTNAM: .ASCII   /NXTFRM/                  ; ASCII named data field name
LIBNAM: .ASCIZ   /DEMLIB/                  ; ASCIZ library name
FILE:   .ASCII   /FILE  /                  ; 6 BYTE ASCII file name
MSG1:   .ASCIZ   /Illegal choice/          ; Message for illegal menu choice
MSG2:   .ASCIZ   *Fatal I/O error*         ; Message with embedded '/'
        .EVEN
;
; Argument lists and data area
;

ARGLST: .BLKB    F$ASIZ                    ; Form Driver argument list
REQLST: .BLKB    F$RSIZ                    ; Form Driver required list
STAT:   .BLKW    2                         ; Form Driver status block

VAR1:   .BLKB    6                         ; Variable 6-byte block for
                                           ; general use

OUTFIL: .BLKW    5                         ; Output file name
FRMNAM: .BLKW    3                         ; Area for form names
SAVNAM: .BLKW    3                         ; Save area for a form name
FILBLK: .BLKW    1                         ; Length of file in blocks
BUFADR: .BLKW    1                         ; Length of data in buffer
BUFFER: .BLKW    256.                      ; Length of output buffer
ENDBUF=.
FILNAM: .BLKB    20                        ; 16 bytes for ASCII file name
IMPUR1: .WORD    ISIZ                      ; First Impure area - for First form
        .BLKB    ISIZ-2
IMPUR2: .WORD    ISIZ                      ; Second Impure area - for Update form
        .BLKB    ISIZ-2
IMPUR3: .WORD    ISIZ                      ; Third Impure area - for Last form
        .BLKB    ISIZ-2
        .EVEN
;
; I/O section
;

OUTFDB: FDBDF$                             ; Define the FDB
        FDAT$A   R.VAR,FD.CR               ; Variable length records with CR/LF
        FDRC$A   ,BUFFER                   ; Allow read, write and modify
        FDOP$A   OU$CHN,DSDS,NMBLK         ; File descriptors

NMBLK:  NMBLK$   ,DAT,,SY,0                ; Default for file

DSDS:   .WORD    0,0                       ; Default the device name
        .WORD    0,0                       ; and file directory
        .WORD    0,0                       ; This is the file name length
                                           ; and file name
        .EVEN

        .SBTTL    MACDEM - FMS Demonstration Subroutine
```

```
;++
; FUNCTIONAL DESCRIPTION:
;
;       This is the MACRO demonstration program for FMS
;       illustrating a simple form-driven, data-entry
;       application.
;--

        .PSECT  MACDEM,I,RO,REL


DEMO:

        FINIT$                          ; Initialize for FCS I/O
        QIOW$S  #IO.ATT,#T$LUN,#T$EFN   ; Attach the terminal
        BCC     1$                      ; If error then just leave
        CALL    LEAVE                   ; Done for now

1$:     MOV     #ARGLST,R0              ; R0 = addr of FDV arg list

        MOV     #REQLST,R1              ; R1 = addr of FDV required arg list
        MOV     #STAT,F$STS(R1)         ; Set addr of status block
        MOV     #IN$CHN,F$CHN(R1)       ; Set I/O channel for FDV
        MOV     #IMPUR1,F$IMP(R1)       ; Set addr of FDV impure area

        $FDV    REQ=R1                  ; Init required arg list pointer
        $FDV    FNC=OPN,NAM=#LIBNAM     ; Open form library
        CALL    ERREX                   ; Exit if error

FIRST:  $FDV    FNC=CSH,NAM=#FSTNAM     ; Show FIRST menu
        CALL    ERREX                   ; Exit if error

        MOV     #IMPUR3,REQLST+F$IMP    ; Reset to THIRD impure area
        $FDV    ARG=#ARGLST,FNC=SHO,REQ=#REQLST,NAM=#LSTNAM,NUM=#19. ;Show LAST
        CALL    ERREX                   ; Exit if error


10$:    MOV     #ARGLST,R0              ; Reset R0 = addr of FDV arg list
        $FDV    FNC=CIA,VAL=#IMPUR1     ; Reset to FIRST impure area

11$:    $FDV    FNC=GET,NAM=#CHCNAM     ; Get field 'CHOICE'
        CALL    ERREX                   ; Exit if error

        MOV     #VAR1,R1                ; R1 = ptr to 6-byte block
        CALL    BLKNAM                  ; Blank out VAR1
        MOVB    @F$VAL(R0),VAR1         ; VAR1 = menu choice
        $FDV    FNC=DAT,NAM=#VAR1       ; Get named data with the name being
                                        ;   the response to 'CHOICE'
        TST     STAT                    ; Was get successful?
        BGT     20$                     ; Continue if ok
        $FDV    FNC=LST,VAL=#MSG1,LEN=#-1 ; Else print message on line 24
        BR      11$                     ; Try again

20$:    MOV     F$VAL(R0),R1            ; R1 = addr of name from named data
        MOV     #EXTNAM,R2              ; R2 = addr of exit name
        CALL    CMPNAM                  ; Zero set on match
        BNE     30$                     ; Continue on match
        JMP     LIBCLS                  ; Else close form library and exit

30$:    CALL    MOVNAM                  ; Save named data
        MOV     #FRMNAM,R1              ; R1 = adr of source name
        MOV     #SAVNAM,R2              ; Adr to save form name
        .REPT   3
            MOV  (R1)+,(R2)+            ; Save form name
        .ENDR
        CLR     FILBLK                  ; Init file length
        MOV     #BUFFER,BUFADR          ; Init buffer adr
        MOVB    VAR1,FILE+4             ; File name = "FILE"+contents of VAR1
        MOV     #FILE,R1                ; R1=addr of 6 byte block holds filename
        $FDV    FNC=DAT,NAM=R1          ; Get named data at VAR1

        CALL    DATSET                  ; Go set up a data set descriptor
        OPEN$W  #OUTFDB                 ; Open the file
        BCC     60$                     ; Continue if ok
        CALL    LEAVE                   ; Leave on I/O error

60$:    $FDV    ARG=#ARGLST,FNC=CIA,VAL=#IMPUR2 ; Select impure area
                                                ; for UPDATE
        $FDV    FNC=SHO,NAM=#FRMNAM,NUM=#7.
        CALL    ERREX                   ; Exit if error

        $FDV    FNC=ALL                 ; Get all data from form
        CALL    ERREX                   ; Exit if error

        CALL    SAVDAT                  ; Put data in file

        $FDV    ARG=#ARGLST,FNC=DAT,NAM=#NXTNAM ; Get name of next form
        CALL    MOVNAM                  ; Put form name in FRMNAM
        MOV     #NONNAM,R1              ; R1 = adr of ASCII .NONE.
        MOV     #FRMNAM,R2              ; R2 = adr of returned name
        CALL    CMPNAM                  ; Zero set on match
        BEQ     70$                     ; Display last form on match
        BR      60$                     ; Else get data from next form
```

```
70$:    $FDV    FNC=CIA,VAL=#IMPUR3     ;Select 3rd impure area for LAST menu
80$:    $FDV    FNC=GET,NAM=#CHCNAM
        CALL    ERREX                   ; Exit if error
        MOV     F$VAL(R0),R1            ; R1 = adr of answer
        CMPB    (R1),#'2                ; Was 2 selected?
        BNE     85$                     ; IF NO, see if 1 was selected
        CLOSE$  #OUTFDB                 ; IF YES Close current file
        MOV     #REQLST,R1              ; Reset pointer to REQLST
        MOV     #IMPUR1,F$IMP(R1)       ; Select 1st impure area
        JMP     10$                     ; Return to 10$, show FIRST menu

85$:    CMPB    (R1),#'1                ; Was 1 selected?
        BNE     90$                     ; 3 was selected
        MOV     #SAVNAM,R1              ; R1 = source name
        MOV     #FRMNAM,R2              ; R2 = dest name
        .REPT   3
        MOV     (R1)+,(R2)+             ; Move name
        .ENDR
        BR      60$

90$:    MOVB    (R1),VAR1               ; Move into variable for name
        MOVB    #40,VAR1+1              ; Make 2nd char blank
        $FDV    FNC=DAT,NAM=#VAR1       ; Get named data
        TST     STAT                    ; Check status
        BGT     CHKCLS                  ; If ok then close file
        $FDV    FNC=LST,VAL=#MSG1,LEN=#-1 ; Print message on line 24
        BR      80$                     ; Try again

;
; Close the output file
;

CHKCLS: CLOSE$  #OUTFDB                 ; Close output file
        MOV     #EXTNAM,R1              ; Name of exit named data
        MOV     #ARGLST,R0              ; Get ARGLST
        MOV     F$VAL(R0),R2            ; R2 = adr of named data
        CALL    CMPNAM                  ; Zero set if match
        BEQ     LIBCLS                  ; Exit on match
        JMP     FIRST                   ; Back to start on no match

LIBCLS: $FDV    FNC=CSH,NAM=#CLRSCR     ; Show menu form
        $FDV    FNC=CLS                 ; Close form library
        BR      EXIT                    ; And exit
;
; Routine to check for error return from Form Driver.
; Print message and exit on error.
;
```

```
ERREX:  TST     STAT                    ; Was call ok?
        BLE     LEAVE
        RETURN

LEAVE:  $FDV    ARG=#ARGLST,FNC=LST,VAL=#MSG2,LEN=#-1
                                        ; Print message on line 24
        $FDV    FNC=CLS                 ; Close form library

EXIT:   EXIT$S

;
; Subroutine to store data in output file
;

SAVDAT: MOV     F$VAL(R0),R2            ; R2 = adr of data returned
        MOV     F$LEN(R0),R3            ; R3 = length of data returned
        TST     R3                      ; Was data returned?
        BLE     10$                     ; If not return
        PUT$    #OUTFDB,2,R3            ; Store away the string of data
10$:    RETURN

;
; Subroutine to move name and blank fill to 6 chars
;       F$VAL(R0) = Addr of source name
;       F$LEN(R0) = Length of source name
;       FRMNAM = Addr of destination of name
;

MOVNAM:
        MOV     #FRMNAM,R1              ; R1 = addr to store form name
        CALL    BLKNAM                  ; Blank out name
        MOV     F$VAL(R0),R1            ; R1 = addr of named data
        MOV     #FRMNAM,R2              ; R2 = addr to store form name
        MOV     F$LEN(R0),R3            ; Length of named data
10$:    MOVB    (R1)+,(R2)+             ; Move named data to form name
        DEC     R3                      ; Dec char ctr
        BNE     10$
        RETURN

;
; Subroutine to blank 6 bytes
;       R1 = Addr of name to blank
;

BLKNAM:
        MOV     #6,R2                   ; R2 = 6
5$:     MOVB    #40,(R1)+               ; Init name with blanks
        DEC     R2                      ; Dec byte ctr
        BNE     5$
        RETURN
```

```
;
; Subroutine to compare two 6-byte names
;       R1,R2 point to names
;       R3 = 0 if match on return
;

CMPNAM:
        MOV     #6,R3                   ; 6 char compare
10$:    CMPB    (R1)+,(R2)+             ; Compare 2 bytes
        BNE     20$                     ; Leave loop if no match
        DEC     R3                      ; Dec char ctr
        BNE     10$
20$:    RETURN
;
; WE ARE GOING TO BUILD THE DATASET DESCRIPTOR BY HAND
;
DATSET:
        MOV     F$LEN+ARGLST,R4         ; Set string size in R4
        MOV     #DSDS,R3                ; Clear the dataset descriptor
        .REPT   6                       ; So we don't imply any defaults
        CLR     (R3)+
        .ENDM
        MOV     F$VAL+ARGLST,R1         ; -> filespec
30$:    MOV     R1,R2                   ; Save -> start of substring
        MOV     #2,R3                   ; Assume its a filename (offset/4)
40$:    TST     R4                      ; End of filespec??
        BEQ     60$                     ; Yes - save filename string
        CMPB    (R1),#']                ; Directory spec?
        BEQ     50$                     ; Yes
        DEC     R4                      ; Decrement the string count
        CMPB    (R1)+,#':               ; Device spec?
        BNE     40$                     ; Nope
        DEC     R1                      ; Backup to take :
        ASR     R3                      ; Adjust for device
50$:    ASR     R3                      ; Adjust for directory
        INC     R1                      ; Adjust for trailing character (: ])
60$:    ASL     R3                      ; Adjust offset in range (0-10)
        ASL     R3                      ; Into dataset descriptor
        ADD     #DSDS,R3                ; Make address
        MOV     R1,(R3)                 ; Make count
        SUB     R2,(R3)+                ; As end-start
        MOV     R2,(R3)                 ; Save address next
        TST     R4                      ; See if done?
        BEQ     70$                     ; Yes
        BR      30$                     ; Nope back for next segment
70$:    RETURN                          ; Return to caller

        .END    DEMO
```

Appendix D
Task-Building FMS Sample Applications

This appendix contains source listings of the command files used to build the FMS demo programs.

```
*************************************************************************
;
;
; BASRMSCLS.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;   Command file to build BASIC+2 demo with FMS and RMS clustered
;
;
;   Underlined items are changes from file built with
;   BASIC+2 BUILD command.
;
BASRMSCLS/CP,BASRMSCLS/-SP=BASRMSCLS/MP
;                  ------------------- Add map file if desired.
;
CLSTR=FDVRMS,RMSRES:RO
;
UNITS = 14
ASG = TI:13:5
;          --         Assign LUN 5 as terminal for FDV.
ASG = SY:6:7:8:9:10:11:12
;         -           Remove LUN 5 from this line.
//
;
;
;************************************************************************
;
;
; BASRMSCLS.ODL
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;   TKB Overlay Description File for BASIC+2 demo
;   clustered with RMSRES resident library
;
;
;   Underlined items are changes from file built by
;   BASIC+2 BUILD command.
;
                    .ROOT BASIC2-RMSROT-USER
;
USER:               .FCTR BASDEM-LIBR-FORM
;                    -----  Add FMS factor to root
LIBR:               .FCTR LB:[1,1]BP2OTS/LB
;
FORM:               .FCTR HLLBP2-HLLDFN-FRM1
;------------------------------- Add HLL interface
FRM1:               .FCTR FDVDRT
;-------------------   Add FDV data area
;
@LB:[1,1]BP2IC1
@LB:[1,1]RMSRLX
                  .END
;
;
```

Figure D-1. Command File and ODL File to Build BASIC-PLUS-2 Demo with FMS and RMS Clustered

```
*******************************************************************
;
;
; BASRMSRES.CMD
;
;                              Copyright (C) 1985 by
;                       Digital Equipment Corporation, Maynard, MA
;
;   Command file to build BASIC+2 demo with RMSRES resident library
;
;   Underlined items are changes from file built with
;   BASIC+2 BUILD command.
;
BASRMSRES/CP,BASRMSRES/-SP=BASRMSRES/MP
;                  ------------------- Add map file if desired.
UNITS = 15
ASG = TI:13:5
;            --        Assign LUN 5 as terminal for FDV.
ASG = SY:6:7:8:9:10:11:12
;        -             Remove LUN 5 from this line.
EXTTSK =  952
;
RESLIB=LB:[1,1]RMSRES/RO
;
//
;
;*******************************************************************
;
;
; BASRMSRES.ODL
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                    TKB Overlay Description File for BASIC+2 demo
;                    with RMSRES resident library
;
;
;                    Underlined items are changes from file built by
;                    BASIC+2 BUILD command.
;
                     .ROOT BASIC2-RMSROT-USER
;                         Add RMS symbols
USER:                .FCTR BASDEM-LIBR-FORM
;                          ----- Add FMS factor to root
LIBR:                .FCTR LB:[1,1]BP2OTS/LB
;
FORM:                .FCTR HLLBP2-HLLDFN-FRM1
;------------------------------- Add HLL interface
FRM1:                .FCTR FDVLRM/LB
;----------------------   Add FDV
;
@LB:[1,1]BP2IC1
@LB:[1,1]RMSRLX
                     .END
;
```

Figure D-2. Command File and ODL File to Build BASIC-PLUS-2 Demo with RMSRES Resident Library

```
************************************************************
;
;
; BASRMSTKB.CMD
;
;                        Copyright (C) 1985 by
;                  Digital Equipment Corporation, Maynard, MA
;
;                 Command file to build BASIC+2 demo
;
;
;                 Underlined items are changes from file built with
;                 BASIC+2 BUILD command.
;
BASRMSDEM/CP,BASRMSDEM/-SP=BASRMSTKB/MP
;               -------------- Add map file if desired.
UNITS = 15
ASG = TI:13:5
;           --            Assign LUN 5 as terminal for FDV.
ASG = SY:6:7:8:9:10:11:12
;       -                 Remove LUN 5 from this line.
EXTTSK =  952
//
;
***************************************************************************
;
; BASRMSTKB.ODL
;
;                        Copyright (C) 1985 by
;                  Digital Equipment Corporation, Maynard, MA
;
;                 TKB Overlay Description File for BASIC+2 demo
;
;
;                 Underlined items are changes from file built by
;                 BASIC+2 BUILD command.
;
.ROOT BASIC2-USER-RMSROT,RMSALL
;
USER:                 .FCTR BASDEM-LIBR-FORM
;                          -----    Add FMS factor to root
LIBR:                 .FCTR LB:[1,1]BP2OTS/LB
;
FORM:                 .FCTR HLLBP2-HLLDFN-FRM1
;------------------------------    Add HLL interface
FRM1:                 .FCTR FDVLRM/LB-LB:[1,1]RMSLIB/LB:RMSSYM
;-------------------------------         Add FDV and define
;                         RMS symbols
@LB:[1,1]BP2IC1
@LB:[1,1]RMS11X
                      .END
;
```

Figure D-3. Command File and ODL File to Build BASIC-PLUS-2 Demo

```
;***********************************************************************
;
;
; C11RMSCLS.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA

;
;                       Command file to build COBOL-11 demo
;
;
C11RMSCLS,C11RMSCLS/-SP=C11DEM
;
;                       COBOL-11 interface and Form Driver library
;
HLLCOB,HLLDFN,FDVDRT
;
;                       COBOL-11 LIBRARY
;
LB:[1,1]COBLIB/LB
;
;                       RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
/
;
CLSTR=FDVRMS,RMSRES:RO
;
UNITS=9
;                       LUN 1 will be the terminal.
ASG = TI:1
;                       LUN 3 will be the output file unit.
ASG = SY:3
;                       Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
//
;
```

Figure D-4. Command File to Build COBOL-11 Demo

```
;******************************************************************
;
;
; C11RMSRES.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                       Command file to build COBOL-11 demo
;        with RMSRES resident library
;
C11RMSRES,C11RMSRES/-SP=C11DEM
;
;                       COBOL-11 interface and Form Driver library
;
HLLCOB,HLLDFN,FDVLRM/LB
;
;                       Run time libraries for COBOL-11, FDV, RMS
;
LB:[1,1]COBLIB/LB
;
;                       RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
;
/
;
UNITS=9
;                       LUN 1 will be the terminal.
ASG = TI:1
;                       LUN 3 will be the output file unit.
ASG = SY:3
;                       Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
;
RESLIB=LB:[1,1]RMSRES/RO
//
;
```

Figure D-5. Command File to Build COBOL-11 Demo with RMSRES Resident Library

```
****************************************************************
;
;
; C11RMSTKB.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                       Command file to build COBOL-11 demo
;
;
C11RMSDEM,C11RMSDEM/-SP=C11DEM
;
;                       COBOL-11 interface and Form Driver library
;
HLLCOB,HLLDFN,FDVLRM/LB
;
;                       Run time libraries for COBOL-11, FDV, RMS
;
LB:[1,1]COBLIB/LB,RMSLIB/LB
/
;
UNITS=9
;                       LUN 1 will be the terminal.
ASG = TI:1
;                       LUN 3 will be the output file unit.
ASG = SY:3
;                       Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
//
;
```

Figure D-6. Command File to Build COBOL-11 Demo

```
;*********************************************************************
;
;
; C81RMSCLS.CMD
;
; Copyright (C) 1979 Digital Equipment Corporation, Maynard, MA
;
;                     Command file to build COBOL-81 demo
;
;
C81RMSCLS,C81RMSCLS/-SP=C81DEM
;
;                     COBOL interface and Form Driver library
;
HLLCOB,HLLDFN,FDVDRT
;
;                     COBOL-81 LIBRARY
;
LB:[1,1]C81LIB/LB
;
;                     RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
;
;
/
;
CLSTR=FDVRMS,RMSRES:RO
;
UNITS=9
;                     LUN 1 will be the terminal.
ASG = TI:1
;                     LUN 3 will be the output file unit.
ASG = SY:3
;                     Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
//
;
```

Figure D-7. Command File to Build COBOL-81 Demo

```
*****************************************************************
;
;
; C81RMSRES.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
; Command file to build COBOL-81 demo with RMSRES resident library
;
;
C81RMSRES,C81RMSRES/-SP=C81DEM
;
;                    COBOL interface and Form Driver library
;
HLLCOB,HLLDFN
;
;                    Run time libraries for COBOL-81, FDV, RMS
;
LB:[1,1]C81LIB/LB,SY:[30,10]FDVLRM/LB
;
;                    RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
;
/
UNITS=9
;                    LUN 1 will be the terminal.
ASG = TI:1
;                    LUN 3 will be the output file unit.
ASG = SY:3
;                    Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
;
RESLIB=LB:[1,1]RMSRES/RO
;
//
;
```

Figure D-8. Command File to Build COBOL-81 Demo with RMSRES Resident Library

```
**************************************************************
;
;
; C81RMSTKB.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                    Command file to build COBOL-81 demo
;
;
C81RMSDEM,C81RMSDEM/-SP=C81DEM
;
;                    COBOL interface and Form Driver library
;
HLLCOB,HLLDFN
;
;                    Run time libraries for COBOL-81, FDV, RMS
;
LB:[1,1]C81LIB/LB,SY:[30,10]FDVLRM/LB,LB:[1,1]RMSLIB/LB
/
UNITS=9
;                    LUN 1 will be the terminal.
ASG = TI:1
;                    LUN 3 will be the output file unit.
ASG = SY:3
;                    Reassign the logical unit in the Form Driver.
GBLDEF = T$LUN:1
//
;
```

Figure D-9. Command File to Build COBOL-81 Demo

```
;************************************************************************
;
;
; DBLRMSCLS.CMD
;
;                            Copyright (C) 1985 by
;                    Digital Equipment Corporation, Maynard, MA
;
;  Command file to build the DIBOL version of the FMS demo with RMS
;
DBLRMSCLS/CP,DBLRMSCLS/CR/-SP=DBLRMSCLS/MP
;
UNITS=18
;ASG=NL0:5            ; Set to null channel
;ASG=TI:16            ; Foreground terminal
;ASG=TI:17            ; DDT terminal
;ASG=SY0:18           ; Channel for RENAM/DELET
CLSTR=DBLRSX,FDVRMS,RMSRES:RO
;
GBLDEF=CHNLST:1042
//
;
;************************************************************************
; DBLRMSCLS.ODL
;
;                            Copyright (C) 1985 by
;                    Digital Equipment Corporation, Maynard, MA
;
;
.ROOT PROG-DBLLIB-HLL-FDVLIB-RMS
PROG:                   .FCTR DBLDEM
DBLLIB:                 .FCTR LB:[1,1]DBLUESL/LB-LB:[1,1]DBLOSSL/LB
HLL:                    .FCTR HLLDBL-DBLDFN
FDVLIB:                 .FCTR FDVDRT
RMS:                    .FCTR RMSROT
@LB:[1,1]RMSRLX.ODL
                        .END
```

Figure D-10. Command File to Build the DIBOL Version of the FMS Demo with RMS

```
*************************************************************
;
;
; DBLRMSRES.CMD
;
;                        Copyright (C) 1985 by
;                  Digital Equipment Corporation, Maynard, MA
;
;
;   Command file to build the DIBOL version of the FMS demo
;
;
DBLRMSRES/CP,DBLRMSRES/CR/-SP=DBLRMSRES/MP
;
UNITS=18
;ASG=NL0:5            ; Set to null channel
;ASG=TI:16            ; Foreground terminal
;ASG=TI:17            ; DDT terminal
;ASG=SY0:18           ; Channel for RENAM/DELET
CLSTR=DBLRSX,RMSRES:RO
;
GBLDEF=CHNLST:1042
//
;

*************************************************************

; DBLRMSRES.ODL
;
;                        Copyright (C) 1985 by
;                  Digital Equipment Corporation, Maynard, MA
;
;
                    .ROOT PROG-DBLLIB-HLL-FDVLIB-RMS
PROG:               .FCTR DBLDEM
DBLLIB:             .FCTR LB:[1,1]DBLUESL/LB-LB:[1,1]DBLOSSL/LB
HLL:                .FCTR HLLDBL-DBLDFN
FDVLIB:             .FCTR FDVLRM/LB
RMS:                .FCTR RMSROT
@LB:[1,1]RMSRLX.ODL
                    .END
```

Figure D-11. Command File to Build the DIBOL Version of the FMS Demo (Page 1 of 9)

```
*************************************************************

;
; F77FCSTKB.CMD
;
;                          Copyright (C) 1985 by
;
;                    Digital Equipment Corporation, Maynard, MA
;
;                    Build the FORTRAN demo for FORTRAN-77 using FCS
;
;
F77FCSDEM/FP,F77FCSDEM/CR/-SP=F77DEM
;
;                    FORTRAN interface and Form Driver library (using FCS)
;
HLLFOR,HLLDFN,FDVLIB/LB
;
LB:[1,1]F77FCSOTS/LB
//
;
```

Figure D-11. Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 2 of 9)

```
****************************************************************
;
; F77RMSCLS.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;  Build the FORTRAN demo for FORTRAN-77 clustered with RMSRES
;
F77RMSCLS/FP,F77RMSCLS/CR/-SP=F77DEM
;
;                   FORTRAN interface and Form Driver library (using RMS)
;
HLLFOR,HLLDFN,FDVDRT
;
LB:[1,1]F77RMSOTS/LB
;
;                   RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
;
/
;
CLSTR=FDVRMS,RMSRES:RO
//
;
```

Figure D-11. Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 3 of 9)

```
*************************************************************
;
; F77RMSRES.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
; Build the FORTRAN demo for FORTRAN-77
;   using RMS resident library
;
F77RMSRES/FP,F77RMSRES/CR/-SP=F77DEM
;
; FORTRAN interface and Form Driver library (using RMS)
;
HLLFOR,HLLDFN
;
FDVLRM/LB,LB:[1,1]F77RMSOTS/LB
;
;                  RMS-11 Resident library modules
;
LB:[1,1]RMSLIB/LB:R0AUTL:R0IMPA:RMSSYM:R0EXSY
;
/
RESLIB=LB:[1,1]RMSRES/RO
//
;
```

Figure D-11. Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 4 of 9)

```
*****************************************************************
;
; F77RMSTKB.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                  Build the FORTRAN demo for FORTRAN-77 using RMS
;
F77RMSDEM/FP,F77RMSDEM/CR/-SP=F77DEM
;
;                  FORTRAN interface and Form Driver library (using RMS)
;
HLLFOR,HLLDFN
;
FDVLRM/LB,LB:[1,1]F77RMSOTS/LB,LB:[1,1]RMSLIB/LB
;
;CLSTR=FDVRMS,RMSRES:RO
//
;
```

Figure D-11. **Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 5 of 9)**

```
*************************************************************************
;
; FORFCSTKB.CMD
;
;                         Copyright (C) 1985 by
;                    Digital Equipment Corporation, Maynard, MA
;
;                 Build the FORTRAN demo
;
;
FORFCSDEM,FORFCSDEM/CR/-SP=FORDEM
;
;                 FORTRAN interface and Form Driver library
;
HLLFOR,HLLDFN,FDVLIB/LB
;
LB:[1,1]FOROTS/LB
//
;
```

Figure D-11. Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 6 of 9)

```
***********************************************************

;
; MACFCSTKB.CMD
|
;
;
;                          Copyright (C) 1985 by
;                   Digital Equipment Corporation, Maynard, MA
;
;                  Task Build the Macro version of the DEMO application
;
;
MACFCSDEM,MACFCSDEM/-SP=MACDEM,FDVLIB/LB
/
UNITS=7
MAXBUF=512
//
;
```

Figure D-11. Command File to Build the DIBOL Version of the FMS Demo (Page 7 of 9)

```
****************************************************************
;
; UTPF77TKB.CMD
;
; Copyright (C) Digital Equipment Corporation, Maynard, MA
;
;   Task Build the FORTRAN-77 version of the UETP application
;   with the script processor
;
;
SISF77/FP,SISF77/-SP=SISF77,SCRIPT,UTPTIO
HLLFOR,HLLDFN,FDVLIB/LB,LB:[1,1]F77FCSOTS/LB
/
MAXBUF=512
//
;
```

Figure D-11. Command File to Build the ~~DIBOL~~ FORTRAN Version of the FMS Demo (Page 8 of 9)

```
****************************************************************

;
; UTPFORTKB.CMD
;
; Copyright (C) 1985 Digital Equipment Corporation, Maynard, MA
;
;                    Task Build the FORTRAN-IV version of the UETP application
;                    with the script processor
;
;
SISFOR,SISFOR/-SP=SISFOR,SCRIPT,UTPTIO
HLLFOR,HLLDFN,FDVLIB/LB,LB:[1,1]FOROTS/LB
/
MAXBUF=512
//
;

*****************************************************************
```

Figure D-11. Command File to Build the DIBOL Version of the FMS Demo (Page 9 of 9)

Appendix E
FMS-11 Software Error Messages

This appendix provides general information about FMS-11 software messages and lists all the diagnostic messages FMS software can produce.

Section C.2 describes the order of messages.

Section C.3 summarizes the types of messages each FMS software component can produce.

Section C.4 explains the procedure to follow if FMS software malfunctions.

Section C.5 includes messages for the Form Editor (FED), the Form Utility (FUT), and the Form Driver (FDV). The messages, arranged alphabetically, are printed in the same form displayed on your terminal.

E.1 How to Use This Appendix

When you receive an FMS message, look it up in Section C.5, read the explanation about the reason for the message, and apply the remedies described.

Often, the Form Editor and the Form Driver signal you with the bell signal on your terminal. In some cases, the bell signal is accompanied by a displayed message, and in some cases there is no explicit message available. Section C.3 summarizes the ways the Form Editor and Form Driver use bell signals. Refer to that section when you hear a bell signal and there is no accompanying message.

E.2 Order of Messages and Special Features

The messages in Section C.5 have been alphabetized according to the following convention:

If the first character in a message is neither a digit nor a letter, the second character is used for alphabetizing.

This appendix uses three general references in messages to stand for specific names or values that are copied directly from the work you are doing. These are:

AAAAAA	A name, such as a form name, that FMS software copies into messages.
NNNNNN	A specific value, such as a number of blocks or bytes, that FMS software reports.
n	A single digit, such as a message code, that FMS provides to distinguish different causes or contexts for a message.

If you have trouble finding a message in Section C.5, review the following procedure:

1. Identify the message's origin. This appendix documents only the messages that apply to FMS software.
2. If the first character in a message is a special character, such as a question mark (?), ignore that character.
3. Ignore any number or name in a message that relates specifically to your program or files.
4. Look up the message using the characters that remain.

E.3 Types of FMS Messages

Each FMS software component uses a special set of conventions in the messages it can produce. This section summarizes those conventions.

E.3.1 Types of Form Editor Messages

The Form Editor can produce the following types of messages:

- Messages that follow your response to the Form Editor's prompt are one line long and are followed by the prompt.
- Messages that occur as you are editing a form appear on the bottom screen line. Pressing the RETURN key removes the message from the screen. You can then continue editing.
- Messages that occur after you finish editing a form appear on the bottom screen line and are followed by the prompt.
- Messages about invalid field entries appear on the bottom screen line. You can continue editing when the cursor reappears within the form.

- When you press the HELP key to get information about a field in a questionnaire, messages one line long appear on the bottom screen line. Pressing the HELP key again will display information about the entire questionnaire.
- When you use an invalid cursor control function, the Form Editor signals you by ringing the bell on your terminal. No message is printed. The bell message signals are summarized later in this section.

E.3.2 Types of Form Utility Messages

Form Utility messages follow your response to the prompt using the following form:

```
?FUT
```

E.3.3 Types of Form Driver Messages

When you are running the Form Driver without debug mode support, the Form Driver can produce only the following types of messages:

- Messages about invalid field entries appear on the bottom screen line, and you can immediately continue editing when the cursor reappears within the form.
- Messages that are one line long appear on the bottom screen line when you press the HELP key to ask for help about a field in a form. HELP for the entire form is displayed if you press the HELP key again.
- The Form Driver signals you by ringing the bell on your terminal when you use an invalid cursor control function but no message is printed. Bell message signals are summarized later in this section.

When you are running the Form Driver with debug mode support, each of the preceding types of messages can also appear, as well as special debug mode messages in the following forms:

- ?FDV-F-Text (for fatal errors).
- ?FDV-W-Text (for warnings).

Each of the Form Driver's debug mode messages has a corresponding status code that is returned to the calling task. The documentation for each debug message includes both the MACRO-11 form and high-level language form of the status code. For example, documentation for the message "?FDV-F-INVALID FIELD SPECIFICATION" includes the following information:

```
Return code:   FE$FLD Value:   (-11)
```

When you have debugged a task and are running it without debug mode support, the task should include code to test for an error on each Form Driver call by checking the status code. MACRO-11 tasks should test for the return code, FE$FLD, in the preceding example. BASIC-PLUS-2 and FORTRAN IV tasks should test for the value returned by the FSTAT call, −11 in the preceding example.

The Form Driver can also return five additional status codes that have no corresponding debug mode messages. The codes and their meanings are:

Code	Value	Meaning
FS$SUC	(1)	Successful completion.
FS$INC	(2)	Current form incomplete.
(none)	(−20)	Wrong number of arguments for a FORTRAN call.
(none)	(−21)	Impure area not initialized for a FORTRAN call.
(one)	(−22)	Output string length too short for BASIC-PLUS-2 call.

E.3.4 Bell Message Signals

The Form Editor and the Form Driver use the bell on your terminal in slightly different ways.

E.3.4.1 Bell Message Signals from the Form Driver — In the debug mode, the Form Driver displays each debug mode message, rings the bell, and then waits for you to press the ENTER key or the RETURN key before continuing. In the run mode, the Form Driver does not display the debug mode messages, does not wait, and does not use the bell signal; it returns directly to the calling task.

In the normal run mode (when the debug mode is disabled), the Form Driver uses the bell to warn the terminal operator about two kinds of input errors: errors within fields, and errors in moving the cursor or in terminating fields.

For typing errors within fields, the Form Driver also displays a short message on the bottom line of the screen. For example, when a field description requires numeric characters and the operator types a letter, the Form Driver rings the bell and displays the message NUMERIC REQUIRED on the bottom line of the screen.

For errors in moving the cursor or in terminating fields, the Form Driver warns the operator only with the bell. The eight conditions that cause the Form Driver to ring the bell are:

1. When the Form Driver cannot echo a character at the current position.
2. When cursor-left is illegal.
3. When cursor-right is illegal.
4. When delete character is illegal.
5. When changing the current input mode is illegal.
6. When the previous field is illegal as a terminator because the current field is the first field in the form that can accept input.
7. When the next field is illegal as a terminator because the current field is the last field in the form that can accept input.
8. When scroll forward, scroll backward, exit scrolled area downward, and exit scrolled area upward are illegal because the current field is not in a scrolled area.

E.3.4.2 Bell Message Signals from the Form Editor — When you are completing any of the questionnaires that the Form Editor uses, the Form Driver displays the questionnaire forms and handles all terminal I/O. Therefore, bell message signals have the same meanings as for the Form Driver when you are completing a questionnaire. Refer to the preceding section for Form Driver message signals in the Form Driver's normal run mode.

The bell is also used when the Form Driver is not in control. The Form Editor, in edit mode, uses the bell to signal illegal input.

E.4 Suggestions to Follow if FMS Software Malfunctions

If you think the FMS software has malfunctioned, use the following procedure:

1. As accurately as possible, write down the functions, commands, terminal input, and user program processes you used before the messages indicating a malfunction appeared.
2. Save any programs and files you were using.
3. Obtain new copies of the FMS components involved, and try to duplicate the malfunction.
4. If you still think FMS software has malfunctioned, check your hardware or find someone to check it for you.
5. If the problem persists, consult someone in your area who is very familiar with FMS software.
6. If you qualify to receive a written reply under DIGITAL's Software Performance Report (SPR) service, follow the directions on the SPR form.

E.5 FMS Software Messages

Alphabetic Required - FDV

Explanation of Message

An alphabetic character is required in the current position. The alphabetic characters are the letters A–Z and a–z, and space. The cursor is immediately repositioned in the field and you can continue typing.

Remedies

Application documentation should include instructions for completing the field.

The cursor is immediately repositioned in the field and you can continue typing.

Alphanumeric Required - FDV

Explanation of Message

An alphabetic or numeric character is required in the current position.

Remedies

Application documentation should include instructions for completing the field.

Arrays not allowed in scrolled area - FED

Explanation of Message

This message indicates that the user has assigned the indexed attribute to a field on a scrolled line. This is not allowed.

Remedies

Press the RETURN key to remove the message from the screen; the Form Editor then places the cursor in the first field of the questionnaire. Answer N for the indexed attribute for each field in the scrolled line. If the scrolling attribute is correct, continue to assign attributes. If any field in the scrolled line is to be an indexed field, exit from the questionnaire, use the NORMAL function to remove the scrolling attribute from the line, and use an ASSIGN command to re-enter and complete the questionnaire. Answer H for the indexed attribute when a field is part of a horizontal array, and answer V when the field is part of a vertical array.

Cannot overwrite left margin - FED

Explanation of Message

This message indicates the user has done a delete left of cursor to the beginning of the line. The user is attempting to undelete-left such that the undelete string would go beyond the left hand screen boundary.

Remedies

Before trying the UNDELLINE function again, move the cursor to a character position that has enough blank space to the left for the string you erased.

Cannot overwrite non-blanks at left - FED

Explanation of Message

This message indicates the user has done a delete left of cursor to the beginning of the line. The user is attempting to undelete-left on top of non-blank characters.

Remedies

Before trying the UNDELLINE function again, move the cursor to a character position that has enough blank space to the left for the string you erased.

Cannot overwrite non-blanks at right - FED

Explanation of Message

This message indicates the user has done a delete right of cursor to the end of the line. The user is attempting to undelete-right on top of non-blank characters.

Remedies

Before trying the UNDELLINE function again, move the cursor to a character position that is blank and has enough blank space to the right for the string you erased.

Cannot overwrite right margin - FED

Explanation of Message

This message indicates the user has done a delete right of cursor to the end of the line. The user is attempting to undelete right so the undelete string would go beyond the right hand screen boundary.

Remedies

Before trying the UNDELLINE function again, move the cursor to a character position that has enough blank space to the right for the string you deleted.

Cannot paste over margins or non-blanks or in scrolled area - FED

This message indicates the PASTE operation just attempted by the user has failed. The user must move the cursor to another position in order to complete the PASTE.

Remedies

Press the RETURN key to remove the message from the screen and to restore the area that has a reversed background, if there is one. Then move the cursor to an area that is entirely blank and is large enough for the selection that you want to move.

Clear character '0' required - FED

Explanation of Message

This message indicates that the user assigned the Zero Fill attribute to a field with a non-zero clear character. This is not allowed.

Remedies

If the Zero Fill attribute is correct, change your answer for the Clear Char attribute to '0.' If the clear character you assigned is correct, change your answer for the Zero Fill attribute to N.

COMMAND - FED

Explanation of Message

This message is the Form Editor command prompt that solicits the user for a command.

Remedies

The Form Editor commands are: EDIT, ASSIGN ALL, ASSIGN NEW, ASSIGN FIELD fldnam, FORM, NAME, SAVE, QUIT, and HELP. They are detailed in Chapter 2.

Default too long for field - FED

Explanation of Message

This message indicates the user has assigned a default value to the field that is longer than the field itself. This is not allowed.

Remedies

Press the RETURN key to remove the message from the screen; the Form Editor places the cursor in the first field of the questionnaire. Shorten the default value to match the field length. If the length of the field is correct, continue assigning attributes. If the field is too short, exit from the questionnaire, lengthen the field, and use an ASSIGN command to re-enter and complete the questionnaire.

Embedded spaces illegal in form name - FED

Explanation of Message

The form name specified contained embedded spaces. This is not allowed.

Remedies

The user must enter a valid form name in the Form-Wide Attributes Questionnaire.

?FED-F-Full Duplex terminal driver required

Explanation of Message

The Form Editor will not work unless the system has the full-duplex terminal driver.

Remedies

RSX-11M/PLUS supports only the full-duplex terminal driver; RSX-11M V3.2 offers the option of the full-duplex terminal driver.

?FED-F-FED Requires VT200 terminal

Explanation of Message

The user's terminal must be a VT200 and have been made known to the system as a VT200.

Remedies

Use the SET command to tell the system the terminal is a VT200:

```
SET /VT200=TI:
```

or,

```
SET /TERM=TI:VT200
```

The *MCR Operation Manual* for RSX has a full description of the SET command.

?FDV-F-ERROR OPENING FORM LIBRARY

Explanation of Message

Return Code: FE$IOL Value: (–4) An error was encountered opening the form library. The I/O error code is returned in the second word of the status block. An error code of zero means that no FDB was available for the library. Otherwise the code is as follows:

For FCS: The code follows the standard for FCS errors as a result of open requests. If the high byte of the word is zero the code is an FCS error code. If the high byte of the word is non-zero, the code is an FCS error code. If the high byte of the word is non-zero the low byte is a directive status code error. The word returned is from F.ERR of the FDB.

For RMS: The error code is the RMS error code returned in the STS word of the FAB after the $OPEN call. ER$LBY is returned if a form library is active on the channel (LUN).

Remedies

Check that the form library file specification is correct and that the file exists on the specified volume. Check that the proper volume is installed and that its device is on-line. If the device directory is corrupt, use a copy of the form library file that is on a working volume. Check that the specified channel number is not currently in use.

?FDV-F-ERROR READING FORM LIBRARY

Explanation of Message

Return Code: FE$IOR Value: (–18) An error was encountered reading the form library. The I/O error code is returned in the second word of the status block.

For FCS: The error code returned is the I/O status block code from a $READ or $CLOSE request on the FDB.

For RMS: This is the STS word of the RAB or FAB. The call was a $READ or $CLOSE to RMS.

Remedies

Check that the volume is installed and that its device is on-line. If the message continues to appear, try another copy of the form library file. If the new copy works, the original copy or its form name directory are corrupt and should be replaced. If the message still continues to appear, refer to the procedures in Appendix C, Section C.5.

?FDV-F-FILE NOT FORM LIBRARY

Explanation of Message

The file specified to open is not a form library. The first word was not RAD50 FLB.

Remedies

Check that the file specification is correct.

?FDV-F-FORM LIBRARY IS NOT OPEN ON CHANNEL

Explanation of Message

Return Code: FE$FCH Value: (–7) For a call to the Form Driver, a form library is not open to the specified channel.

Remedies

Check that an FLOPEN call precedes the request to display a form. If the form library was properly opened, check that the correct channel number was specified to the Form Driver (in the last FLCHAN call for high-level languages and in the F$CHN argument of the Required Argument List for MACRO).

?FDV-F-ILLEGAL FOR DISPLAY ONLY FIELD

Explanation of Message

Return Code: FE$DSP Value: (–13) Input is not allowed in a display-only field. This error is returned for a call to get a field if the specified field is display-only, for a call to get any or all fields if all fields in the form are display-only, and for a call to get any field if the current field is display-only.

Remedies

Check that the field attributes are correct and that the latest FSHOW or FCLRSH call displayed the correct form. If the call that caused the message is an FGET call, check that the call uses the correct field identifier.

?FDV-F-ILLEGAL CALL TO FORM DRIVER

Explanation of Message

Return Code: FE$IFN Value: (–19) The specified function is illegal in the current context. The calls to get all fields, put all fields is only illegal for a form with a scrolled area if data is specified. With no data, it is legal, and to get any field is illegal if the current form contains a scrolled area. Only the calls to open a form library, close a form library, display a form, and to display data on the last line are legal if a form is not currently displayed.

Remedies

If the current form contains a scrolled area, correct the program logic so that illegal Form Driver calls are not executed. If no form has been referenced, check for proper use of the FLOPEN, FSHOW, FCLOSE, FPUTL, and FCLRSH calls and for improper flow of control that skips those calls.

?FDV-F-ILLEGAL FILE SPECIFICATION

Explanation of Message

Return Code: FE$FSP Value: (–3) The file name specified for the form library is not a legal file specification.

For FCS: .PARSE returned an error of some type.

For RMS: $OPEN returned one of the following error codes: ER$DEV, ER$DIR, ER$FNM, ER$VER.

Remedies

Correct the file specification.

?FDV-F-IMPURE AREA TOO SMALL

Explanation of Message

Return Code: FE$IMP Value: (–2) The impure area specified is not large enough to allocate the data structures required by the Form Driver to display the form.

Remedies

If the message appears for a FORTRAN IV, BASIC-PLUS-2, COBOL, or FORTRAN IV PLUS task, check that the impure area is at least 64 bytes larger than the largest form the task uses. If the message appears for a MACRO-11 task, check that the F$IMP pointer in the Required Argument List points to an impure area that is at least as large as the largest form the task uses. For MACRO-11 tasks the first word in the impure area must contain the size of the impure area.

?FDV-F-INVALID CALL TO GET NAMED DATA

Explanation of Message

Return Code: FE$DMN Value: (–15) A call to get named data is invalid for one of the following reasons:

- No named data exists for the current form.
- The data name specified does not exist.
- The index specified does not exist.

Remedies

Check for a program error that causes the call to be executed when the current form is the wrong form. Check for the correct name in an FNDATA call and the correct index value in an FIDATA call.

?FDV-F-INVALID CHANNEL NUMBER SPECIFIED

Explanation of Message

Return code: FE$ICH Value: (–6) The channel number specified in a call to the Form Driver is not a valid channel number for the task. Second word of status block contains the I/O status code from system in the event of an error.

For FCS: The error code is from the F.ERR offset of the FDB.

For RMS: The error code is from the STS word of the FAB.

Remedies

Correct the channel number the task is using or specify a different range of channels for your application.

?FDV-F-INVALID FIELD SPECIFICATION

Explanation of Message

Return Code: FE$FLD Value: (–11) The field specified does not exist. An invalid field name or an invalid index for the field was specified.

Remedies

Check the field name or array index. Also check for a program error that causes the call to be executed at the wrong time or for the wrong form.

?FDV-F-INVALID FIRST LINE TO DISPLAY FORM

Explanation of Message

Return Code: FE$LIN Value: (–10) The entire form will not fit on the screen if displayed starting at the line number specified in the call to the Form Driver or the line number is not from 1 to 23 inclusive.

Remedies

Check that the terminal has the features you need and is set properly for the form you want to use. Check for a program error that causes the wrong starting screen line number. For forms that are to be offset on the screen, check that the associated HELP forms are properly designed for the full range of offset positions.

?FDV-F-INVALID FORM DEFINITION

Explanation of Message

Return Code: FE$FRM Value: (–8) The format of the form description is not valid.

Remedies

Refer to the procedures in Appendix C, Section C.5.

?FDV-F-INVALID FUNCTION CODE

Explanation of Message

Return Code: FE$FCD Value: (–1) The function code specified for a call to the Form Driver does not exist.

Remedies

If a MACRO-11 task caused the message to appear, check for a typing error in the function code. If the function code is correct or, if a FORTRAN IV, BASIC-PLUS-2, COBOL, or FORTRAN-77 task caused the message to appear, refer to the procedures in Appendix C, Section C.5.

?FDV-F-NO FIELDS DEFINED FOR FORM

Explanation of Message

Return Code: FE$NOF Value: (–12) Calls pertaining to fields are illegal if no fields are defined for the current form.

Remedies

Check that the form has been designed properly. Also check for a program error that causes the field processing call to be executed for the wrong form. For example, check for consecutive FSHOW calls that display more than one form on the terminal screen simultaneously but display a form that has only constant text last.

?FDV-F-SPECIFIED FIELD NOT IN SCROLLED AREA

Explanation of Message

Return Code: FE$NSC Value: (–14) The name of a field is required to identify the scrolled area the call pertains to. the specified field is not in a scrolled area.

Remedies

Check the current form. If the form is correct and if it has been designed properly, check for a typing error in the field name.

?FDV-F-UNDEFINED FIELD TERMINATOR

Explanation of Message

Return Code: FE$UTR Value: (–17) The field terminator code specified in a call to process a field is less than 0 or greater than 9.

Remedies

Correct the field terminator code.

?FDV-F-UNDEFINED FORM

Explanation of Message

Return Code: FE$FNM Value: (–9) The specified form is not defined.

Remedies

Check that the correct form library file is open and that the channel number specified to the Form Driver is the one specified when the form library file was opened and that the call specifies the correct form name.

?FED-F-Error reading input form file

Explanation of Message

An I/O error was returned from a call to read the input file.

Remedies

Check that the form library volume is installed and that its device is on-line. If the message continues to appear, try another copy of the form library file. If the new copy works, the original copy should be replaced. If the message still continues to appear, refer to the procedures in Appendix C, Section C.5.

?FED-Form being saved - FED

Explanation of Message

This message indicates to the user that the form has begun the process of being written to the output device.

Remedies

The message is for information only. (RT only.)

Form is not in proper format - FED

Explanation of Message

The specified form is not in the proper format and therefore cannot be edited.

Remedies

Check that your input file specification is correct and check for a bad block in the file. Also try using a copy of the file that does work with the Form Editor or Form Driver. If the message continues to appear, refer to the procedures in Appendix C, Section C.5.

?FED-Form not saved on QUIT - FED

Explanation of Message

This message indicates to the user that when a QUIT operation was executed, the form that was being edited was not saved.

Remedies

The message is for information only.

Full Field Required - FDV

Explanation of Message

The current field must be completely filled and contain no fill characters.

Remedies

Application documentation should include instructions for completing the field.

?FED-Illegal command line - FED

Explanation of Message

The syntax of the command line entered in response to the `FED>` prompt was invalid.

Remedies

Retype the command line correctly. Use /IO to get the version number; /CR to create a new form.

?FED-Input file is not form file or form library - FED

Explanation of Message

The specified input file is not a valid form file or form library.

Remedies

Check the input file specification.

?FED-F-Insufficient Memory

Explanation of Message

This message indicates there is not enough memory to create a new form or to edit the specified form.

Remedies

There must be at least twice the memory required for a form description for editing to take place.

?FED-F-Invalid form name

Explanation of Message

The form name was longer than six characters or was all spaces.

Remedies

The form name specified in response to the Form name? prompt (after a form library file was specified as the input file) was not valid as a form name.

?FED-F-Unable to attach terminal

Explanation of Message

The Form Editor is unable to attach the user's terminal, therefore cannot proceed.

Remedies

Check to see if there are too many terminals attached, too many programs running, or if some other task has your terminal attached.

?FED-F-Unable to create output file

Explanation of Message

The specified output file cannot be created.

Remedies

File cannot be created for many possible reasons: not enough disk space, a protection violation, a hard error.

?FED-Unable to open input file - FED

Explanation of Message

The specified input file does not exist or cannot be opened for editing.

?FED-Write error - output file not saved - FED

Explanation of Message

This message indicates that there was an error writing to the output file and that the form was lost.

Remedies

Check that the output volume is properly installed and that its device is on-line and write-enabled. If a hardware problem has caused the message to appear, contact your DIGITAL service representative.

?FDV-W-DATA TOO LONG

Explanation of Message

Return Code: FE$DLN Value: (–16) The data specified to output is too long. The Form Driver truncates the data and proceeds.

Remedies

Check for program errors that cause the data string to be too long. Check that the form has been designed properly.

?FUT - Clear character invalid Field name == AAAAAA

Explanation of Message

The named field has the Zero Fill attribute, but '0' is not the assigned clear character.

Remedies

See "FUT - Invalid form header format."

?FUT - Command file depth exceeded

Explanation of Message

A third-level indirect command file specifies another indirect command file. Indirect command files for the Form Utility can be nested to a depth of three.

Remedies

Combine two of the indirect command files you tried to use, or revise the set of indirect command files so that no more than three are executing at one time.

?FUT - Command file error unrecognized

Explanation of Message

The indirect command file processor has returned an error code that the Form Utility cannot understand and handle.

Remedies

Confirm that each line in the indirect command files that you are using is valid. If the message appears for a file that you are sure is correct, complete the procedure in Appendix C, Section C.4.

?FUT - Command file I/O error

Explanation of Message

An I/O error occurred when reading an indirect command file. Two causes for this message are:

- A bad I/O device.
- A bad block on an I/O volume.

Remedies

Follow the procedures that have been established to cover possible hardware errors on your system.

?FUT - Command file illegal file specification

Explanation of Message

The specification for an indirect command file is invalid.

Remedies

Examine all indirect command file specifications, including those in nested indirect command files. Check for typing errors and for specifications that are correct on another operating system but not legal on the system you are using.

?FUT - Command file line too long

Explanation of Message

A line in an indirect command file is longer than the 132-character maximum length. A cause for this message is a line that is longer than 132 characters, although each part can be shorter than that.

Remedies

Break the overlength line by repeating its command in separate lines. Or, reorganize the indirect command file.

?FUT - Command file open error

Explanation of Message

The Form Utility could not find a specified indirect command file or found it but could not open it. If the system has locked the indirect command file, the Form Utility displays this message.

Remedies

Check that all indirect command file specifications are correct and complete, and check that the default or explicit volumes are installed and on-line. Use PIP to check for an indirect command file that is locked.

?FUT - Default text too long Field name = AAAAAA

Explanation of Message

The named field has a default value that is longer than the field can display.

See "FUT – Invalid form header format."

?FUT - Display-only field is full or required Field name = AAAAAA

Explanation of Message

The named field has the Display Only attribute and also the Must Fill attribute or the Response Required attribute, a combination of attributes that is invalid.

Remedies

See "FUT - Invalid form header format."

?FUT - Error closing input file

Explanation of Message

The file I/O routines that support the Form Utility detected an error while closing an input file. Two causes for this message are:

- A bad input file device.
- A bad block on an input volume.

Remedies

For each input file, check the directory entry for information about the file being locked or not usable. Also try to make a copy of each input file. If all input files can be copied, retry the Form Utility operation that failed originally. If the Form Utility operation still fails, check the procedure in Appendix C, Section C.4.

?FUT - Error closing or spooling output file

Explanation of Message

The file I/O routines that support the Form Utility detected an error while closing or spooling an output file. Two causes for this message are:

- A bad output or spooling device.
- A bad block on an output volume.

Remedies

Try using a different output device and volume. If the message continues to appear, follow the procedures that have been established to cover possible hardware errors on your system.

?FUT - Error reading input file

Explanation of Message

The file I/O routines that support the Form Utility detected an error while reading an input file.

Remedies

See the causes and suggestions for "FUT - Error closing input file."

?FUT - Error writing output file block

Explanation of Message

The file I/O routines that support the Form Utility detected an error while writing an output file block.

Remedies

See the causes and suggestions for "FUT - Error closing or spooling output file."

?FUT - Error writing record to output file

Explanation of Message

The file I/O routines that support the Form Utility detected an error while writing an output file record.

Remedies

See the causes and suggestions for "FUT - Error closing or spooling output file."

?FUT - Field beyond screen Field name = AAAAAA

Explanation of Message

The named field extends beyond Column 80 and the form has the 80-column attribute.

Remedies

See "FUT - Invalid form header format."

?FUT - Form not in library

Explanation of Message

The input form library file specified in the Form Utility prompt does not contain the form you requested.

Remedies

Use the /LI option with the input form library file name to list the names of all forms in the form library file. Then type another form name.

?FUT - Form AAAAAA replaced

Explanation of Message

The description for the named form has been replaced in the named form library file.

Remedies

The message is for your information only.

?FUT - Illegal command

Explanation of Message

The command line syntax is incorrect. Some causes for this message are:

- More than one output file specified.
- No characters in command line.
- Command line includes both a file specification and the /ID or /HE option.
- Command line with the /FF option includes an output file specification.
- No output file specified for /CR, /DE, /EX, /OB, or /RP options.

Remedies

Correct the command line.

?FUT - Illegal file specification

Explanation of Message

An input or output file specification is incomplete or contains illegal characters. Wildcard characters in a file specification cause this message.

Remedies

Remove any wildcards that are in the file specifications, and check the file specifications for the file names and types that are required for the options you are using.

?FUT - Illegal input file specification or option

Explanation of Message

An input file specification is invalid, or an option is invalid.

Remedies

Check the input file specifications and options for typing errors.

?FUT - Illegal output file or option

Explanation of Message

The output file specification is invalid, or an option is invalid. Some causes for this message are:

- The /SP option is included with another option that produces no printable output.
- The /BA option is included with another option that produces no form library files.
- Options such as /CR and /DE conflict.

Remedies

Check the output file specification for typing errors. Check that the options are consistent with one another.

?FUT - Illegal replacement of form, use /RP

Explanation of Message

The Form Utility command does not include the /RP option but tries to replace a form description that is in the form library file specified in the Form Utility prompt. One cause for this message is combining two form library files that contain a form description with the same name.

Remedies

Use the /LI option to check the names of all forms in the input files. Use the /DE option to delete individual form descriptions. Use the /RP option to combine form library files, and preserve only the last form description processed for each duplicate.

?FUT - In form library = AAAAAA, Form name = AAAAAA

Explanation of Message

This message appears only as the first line of a two-line message.

Remedies

See the description for the other message that the Form Utility has displayed.

?FUT - Insufficient memory

Explanation of Message

Although the Form Utility can run, there is not enough memory to process any form descriptions. In addition to the Form Utility's basic requirements, it requires enough memory for a form library file directory and one form description.

Remedies

Ask your system manager to check whether the Form Utility has been installed as a check-pointable task. The Form Utility should always be check-pointable. Install or run the Form Utility with a larger memory increment.

?FUT - Insufficient space for output buffer

Explanation of Message

See "FUT - Insufficient memory."

?FUT - Invalid field descriptor

Explanation of Message

See "FUT - Invalid form header format."

?FUT - Invalid fixed-decimal picture Field name = AAAAAA

Explanation of Message

The named field has the Fixed Decimal attribute, but the field picture is improper for one of the following reasons:

- There is no decimal point.
- There are two or more decimal points.
- The first and last picture characters are not '9.'

Remedies

See "FUT - Invalid form header format."

?FUT - Invalid form description

Explanation of Message

See "FUT - Invalid form header format."

?FUT - Invalid form header format

Explanation of Message

The Form Utility has detected an error in the form description format and stopped processing the form description. The form with the error might have been corrupted by I/O or device errors and cannot be used with an FMS application.

Remedies

First, try to recover the form description by using an earlier version of a form description file or form library file.

If all versions of the form description cause this message or similar ones, use the following procedure:

1. Retype the form completely with the Form Editor. Do not try to edit a version that causes the message.
2. If problems persist, refer to Appendix C, Section C.4.

?FUT - Invalid form name

Explanation of Message

The last form name you typed contains characters that are not valid. The following examples illustrate possible causes of this message:

- Including a non-Radix-50 character.
- Responding with an asterisk (*) after using the /FF option.

Remedies

Type a valid form name.

?FUT - Invalid index value Field name = AAAAAA

Explanation of Message

The two causes for this message are:

- The named field has the vertical (indexed) attribute and the lowest element of the array is below the last line that is currently assigned for the form.
- The named field is in a line that has the scrolling attribute and the scrolled area is only one line long.

Remedies

See "FUT - Invalid form header format."

?FUT - Invalid named data section

Explanation of Message

See "FUT - Invalid form header format."

?FUT - Invalid number of fields

Explanation of Message

See "FUT - Invalid form header format."

?FUT - Invalid text section

Explanation of Message

See "FUT - Invalid form header format."

?FUT - Logic error - Exception stack overflow

Explanation of Message

Follow the procedure in Appendix C, Section C.4.

FUT - Logic error - Exception stack underflow

Explanation of Message

Follow the procedure in Appendix C, Section C.4.

?FUT - Logic error - pass 2 illegal file number

Explanation of Message

Follow the procedure in Appendix C, Section C.4.

?FUT - Logic error - pass 2 too few input files

Explanation of Message

Follow the procedure in Appendix C, Section C.4.

?FUT - Not a valid form file or library

Explanation of Message

An input file does not contain the proper code words that identify form description files and form library files.

Remedies

Check each input file specification for typing errors, and check for having typed the wrong file name.

?FUT - No forms in library

Explanation of Message

After specifying the /CR, /DE, /EX, or /RP option to create a form library file, you extracted no forms for the output form library file or deleted all forms from the input form library files. The message is a warning that your specified output form library file has not been created.

Remedies

Type a command.

?FUT - Right Justified field with mixed-picture Field name = AAAAAA

Explanation of Message

The named field has both the right-justified attribute and a mixed-picture. The combination is invalid.

Remedies

See "FUT - Invalid form header format."

?FUT - Scrolled and array field Field name = AAAAAA

Explanation of Message

The named field has the horizontal or vertical (indexed) attribute and is also located in a line that has the scrolling attribute.

Remedies

See "FUT - Invalid form header format."

?FUT - Unable to delete output file

Explanation of Message

When a Form Utility process fails, the Form Utility usually deletes the partially complete output file because it is faulty. This message appears when the delete action cannot be completed.

Remedies

See the causes and suggestions for "FUT - Error closing or spooling output file."

?FUT - Unable to open input file

Explanation of Message

See "FUT - Error reading input file."

?FUT - Unable to open output file

Explanation of Message

See "FUT - Error closing or spooling output file."

?FUT - Unable to reopen input file

Explanation of Message

Follow the procedure in Appendix C, Section C.4.

FUT - Zero length field Field name = AAAAAA

Explanation of Message

The named field is zero characters long.

Remedies

See "FUT - Invalid form header format."

Illegal Command - FED

Explanation of Message

This message indicates that the user has typed an illegal command in response to a COMMAND: prompt.

Remedies

Press the RETURN key to remove the message from the screen. The Form Editor then redisplays the COMMAND: prompt. Type in any of the Form Editor commands.

Type HELP to get a list of the valid commands.

Input Required - FDV

Explanation of Message

At least one non-fill character must be entered in the current field.

Remedies

Application documentation should include instructions for completing the field. Press the RETURN key to remove the message from the screen; the Form Editor then displays the cursor in the field you must complete.

Insert line not allowed - FED

Explanation of Message

This message indicates that the user attempted either an Undelete-Line or an Openline when the last line was non-blank.

Remedies

Press the Return key to remove the message from the screen; the Form Editor places the cursor where the function failed. The way you should continue depends on your form. For example, you can erase part of the form below the current line, or raise the text and fields in the form by erasing part of the form above the current line.

Insert not allowed - FED

Explanation of Message

This message indicates that the user attempted to insert a character when the last character on the line was not blank.

Remedies

Press the RETURN key to remove the message from the screen; the Form Editor places the cursor where the insertion failed. The way you should continue depends on your form. For example, you can switch to the OVERSTRIKE mode, or continue in the INSERT mode and erase part of the current line.

Invalid picture for a fixed-decimal field - FED

Explanation of Message

This message indicates that the user has assigned the Fixed Decimal attribute to a field whose picture does not meet the requirements for this attribute.

Remedies

Press the RETURN key to remove the message from the screen; the Form Editor places the cursor at the first field in the questionnaire. Change the answer to the Fixed Dec question to N. If the field is not a fixed-decimal field, continue assigning attributes. If the field should be a fixed-decimal field, exit from the questionnaire, correct the field picture, and use an ASSIGN command to re-enter and complete the questionnaire.

The requirements for a fixed-decimal field are:

- A numeric field.
- Exactly one embedded decimal point - the decimal point cannot be in either the first or last character position in the field.

Logic error - bad field data character - FED

Explanation of Message

This message indicates that an unknown field character was found in a field. If this occurs, the screen image was probably altered by some external disturbance.

Remedies

Run the memory diagnostics and notify your DIGITAL service representative.

New form exceeds available memory - form lost - FED

Explanation of Message

You created a form that was too large for the Form Editor to process with the memory available. The form is lost.

Remedies

There is no way to recover the form.

NO HELP AVAILABLE - FDV

Explanation of Message

No further HELP is available for the current form.

Remedies

Application documentation should include instructions for the operator.

Non-Displayable Character - FDV

Explanation of Message

No validation is required in the current position. However, the character entered cannot be displayed and is therefore invalid.

Remedies

Application documentation should include instructions for completing the field.

Numeric Required - FDV

Explanation of Message

A numeric character (0-9) is required in the current position.

Remedies

Application documentation should include instructions for completing the field.

Only NNNNNN memory blocks left. Continue (Y,N)? - FED

Explanation of Message

This message indicates that only a small amount of memory (the specified number of 512 byte blocks) is available to increase the size of the form. The user is given the option to continue.

Remedies

Type N to cancel the file specification string. The Form Editor then prompts you for another file specification string. Type Y to proceed with the Form Editor session. However, if your editing increases the form size by too much, the Form Editor will not warn you and your editing work will be lost.

Repeat - FED

Explanation of Message

This is the Form Editor REPEAT prompt, which indicates that the user has typed GOLD/"digit."

Remedies

All the following digits will be saved and the first command key typed after the digits will be repeated that number of times.

Right Justified is illegal for a Mixed Picture field - FED

Explanation of Message

This message indicates that the user has assigned the Right Justified attribute to a mixed-picture field. This is not allowed. The user must type a character to continue and the field attributes will be redisplayed and the user must enter "N" to the Right Justified attribute.

Remedies

Press the RETURN key to remove the message from the screen; the Form Editor places the cursor at the first field in the questionnaire. Answer the Right Just question N. If the field should have a mixed picture, continue assigning attributes. If the field should not have a mixed picture, exit from the questionnaire, change the field picture (perhaps by dividing the field into separate fields), and use an ASSIGN command to re-enter and complete the questionnaire.

Signed Numeric Required - FDV

Explanation of Message

A valid signed numeric character (0-9, ".", "–," "+") is required in the current position.

Remedies

Application documentation should include instructions for completing the field.

VIDEO - FED

Explanation of Message

This is the Form Editor video attribute prompt.

Remedies

The user must respond with either a video attribute or press the ENTER key to return to editing.

Index