PRO/FMS-11 Documentation Supplement

Order No. AA-P103C-TK

November 1985

This supplement describes the differences between PRO/FMS-11 and FMS-11/RSX. You can use PRO/FMS-11 with the PRO/Tool Kit or the Professional Host Tool Kit to write applications for the Professional Series computers.

REQUIRED SOFTWARE: Professional Host Tool Kit V3.0, or PRO/Tool Kit V3.0

OPERATING SYSTEM: P/OS V3.0

digital™

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754-2571

First Printing, December 1982
Revised, May 1983
Revised, November 1985

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

Copyright © 1985 by Digital Equipment Corporation
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTI BUS	MASSBUS	Rainbow
DEC	PDP	RSTS
DECmate	P/OS	RSX
DECsystem-10	PRO/BASIC	Tool Kit
DECSYSTEM-20	PRO/Communications	UNIBUS
DECUS	Professional	VAX
DECwriter	PRO/FMS	VMS
DIBOL	PRO/RMS	VT
digital™	PROSE	Work Processor
	PROSE PLUS	

CONTENTS

PREFACE

Manual Objectives

This document describes the differences between FMS-11/RSX applications and PRO/FMS-11 applications. It supplements two manuals: *FMS-11/RSX Software Reference Manual* and *FMS-11/RSX Release Notes*. Both manuals are included in the Tool Kit Documentation Set.

Intended Audience

This document is intended for experienced FMS-11 programmers. If you have not previously programmed with FMS-11, read the *FMS-11/RSX Software Reference Manual*.

Also, you should have some experience with the Professional Host Tool Kit or the PRO/Tool Kit.

Structure of This Document

This document contains seven chapters.

Chapter 1, **Introduction to PRO/FMS-11**, describes the programming languages that you can use to develop a PRO/FMS-11 application. The chapter also illustrates the PRO/FMS-11 development cycle.

Chapter 2, **The Form Editor**, lists the steps you follow to run PROFED in terminal emulation. It also lists attributes that you should avoid when running PRO/FMS-11 applications.

Chapter 3, **The Form Driver**, describes how to use the object module Form Driver. Included are directions for editing the command (.CMD) and descriptor (.ODL) files and linking with the object module.

Chapter 4, **Cluster Libraries**, provides information on using the Form Driver cluster libraries. Included are directions for linking with cluster libraries, and examples that use the cluster libraries.

Chapter 5, **Enhancements to Help Facilities**, explains how to create and use help for PRO/FMS-11.

Chapter 6, **Installing Optional Applications**, tells you how to install the Debug Form Driver and the demonstration library for running the sample programs.

Chapter 7, **Sample PRO/FMS-11 Programs**, lists sample programs for each of the following PRO/Tool Kit languages: BASIC-PLUS-2, COBOL-81, FORTRAN-77, MACRO-11, PASCAL.

Associated Documents

This document supplements two manuals:

- *FMS-11/RSX Software Reference Manual*
- *FMS-11/RSX Release Notes*

These manuals are part of the Tool Kit documentation set.

Note that the *FMS-11/RSX Mini-Reference* is not part of the Tool Kit documentation set, but may be useful when developing PRO/FMS-11 applications.

Conventions Used in This Document

Convention/Term	Meaning
[optional]	In a command line, square brackets indicate that the enclosed item is optional. In a file specification, square brackets are part of the required syntax.
UPPERCASE	Uppercase words and letters, used in examples, indicate that you should type the word or letter exactly as shown.
lowercase	Lowercase words and letters, used in format examples, indicate that you should substitute a word or value of your own. Usually the lowercase word identifies the type of substitution required.

Convention/Term	Meaning
...	A horizontal ellipsis indicates that you can repeat the preceding item one or more times. For example: parameter [,parameter...]
. . .	A vertical ellipsis in a figure or example means that not all of the statements are shown.
Tool Kit	This general term refers to the software you use to develop applications to run on a Professional computer.
Host Tool Kit	The Host Tool Kit is Tool Kit software that runs on a host computer, rather than on the Professional itself.
PRO/Tool Kit	The PRO/Tool Kit is the Tool Kit software that runs on the Professional computer.
red	Interactive input appears in red.

CHAPTER 1

INTRODUCTION TO PRO/FMS-11

PRO/FMS-11 is a development tool based on FMS-11. A forms-oriented video I/O management system, PRO/FMS-11 allows you to develop applications on either the Professional Host Tool Kit (under RSX-11M/11M-PLUS or VAX/VMS) or the PRO/Tool Kit (under P/OS). You can then run the application on your Professional.

This documentation supplement describes the differences between FMS-11 applications designed for minicomputer/VT100 systems and PRO/FMS-11 applications designed for the Professional.

1.1 DEVELOPING PROGRAMS THAT USE PRO/FMS-11

You can develop PRO/FMS-11 applications in the following Tool Kit languages:

- BASIC-PLUS-2
- COBOL-81
- FORTRAN-77
- MACRO-11
- PASCAL

For specific restrictions and sample programs, see Chapter 7.

1.2 PRO/FMS DEVELOPMENT CYCLE

The following sections describe the program development cycle for PRO/FMS-11 applications.

Step 1: **Create FMS Forms**

Create forms on the development system with the PRO/FMS-11 Form Editor, PROFED. To run PROFED, invoke one of the following commands, depending on your development system:

- On RSX (Host Tool Kit), type:

```
>RUN $PROFED
```

- On VMS (Host Tool Kit), type:

```
$ RUN SYS$SYSTEM:PROFED
```

- On P/OS (PRO/Tool Kit), type:

```
$ RUN $PROFED
```

See Chapter 2 for a complete description of PROFED.

Step 2: **Create Form Library**

Create a form library with the PRO/FMS-11 Form Utility, PROFUT. To run PROFUT, invoke one of the following commands, depending on your development system:

- On RSX (Host Tool Kit), type:

```
>RUN $PROFUT
```

- On VMS (Host Tool Kit), type:

```
$ RUN SYS$SYSTEM:PROFUT
```

- On P/OS (PRO/Tool Kit), type:

```
$ RUN $PROFUT
```

Step 3: **Write Source Code**

Write a source program containing the necessary Form Driver calls in the source code.

Step 4: **Compile or Assemble the Program**

Use your compiler or assembler to process the source code.

Step 5: **Modify Command (.CMD) File and Descriptor (.ODL) File**

Choose either the object module Form Driver or cluster library Form Driver. To use the object module Form Driver, follow the instructions in Section 3.2.2. To use the cluster library Form Driver, follow the instructions in Section 4.4.

You should use the debug version of the Form Driver to debug your program on the Professional. Then, when the program is error-free, use the non-debug version.

Step 6: **Task Build the Program**

Use the Professional Application Builder to task build your program. See the *Tool Kit User's Guide* and the *RSX-11M/M-PLUS Task Builder Manual* for details.

Step 7: **Write Installation Command File**

Create the application installation file. See the *Tool Kit User's Guide* for details.

Step 8: **(Host Tool Kit Only) Copy Files to Professional**

If you are using the Host Tool Kit for program development, copy the form library, task image, and installation command file to the Professional.

Step 9: **Install Application Onto P/OS**

Using either Fast Install or P/OS Install Application Services, install the application onto a P/OS menu. See *Tool Kit Reference Manual* for details on Fast Install. See the *User's Guide Hard Disk System* for details on installing an application.

Step 10: **Run the Program**

Choose the installed application from the P/OS menu on which it is installed.

Figure 1-1 illustrates the development cycle with the PRO/Tool Kit.

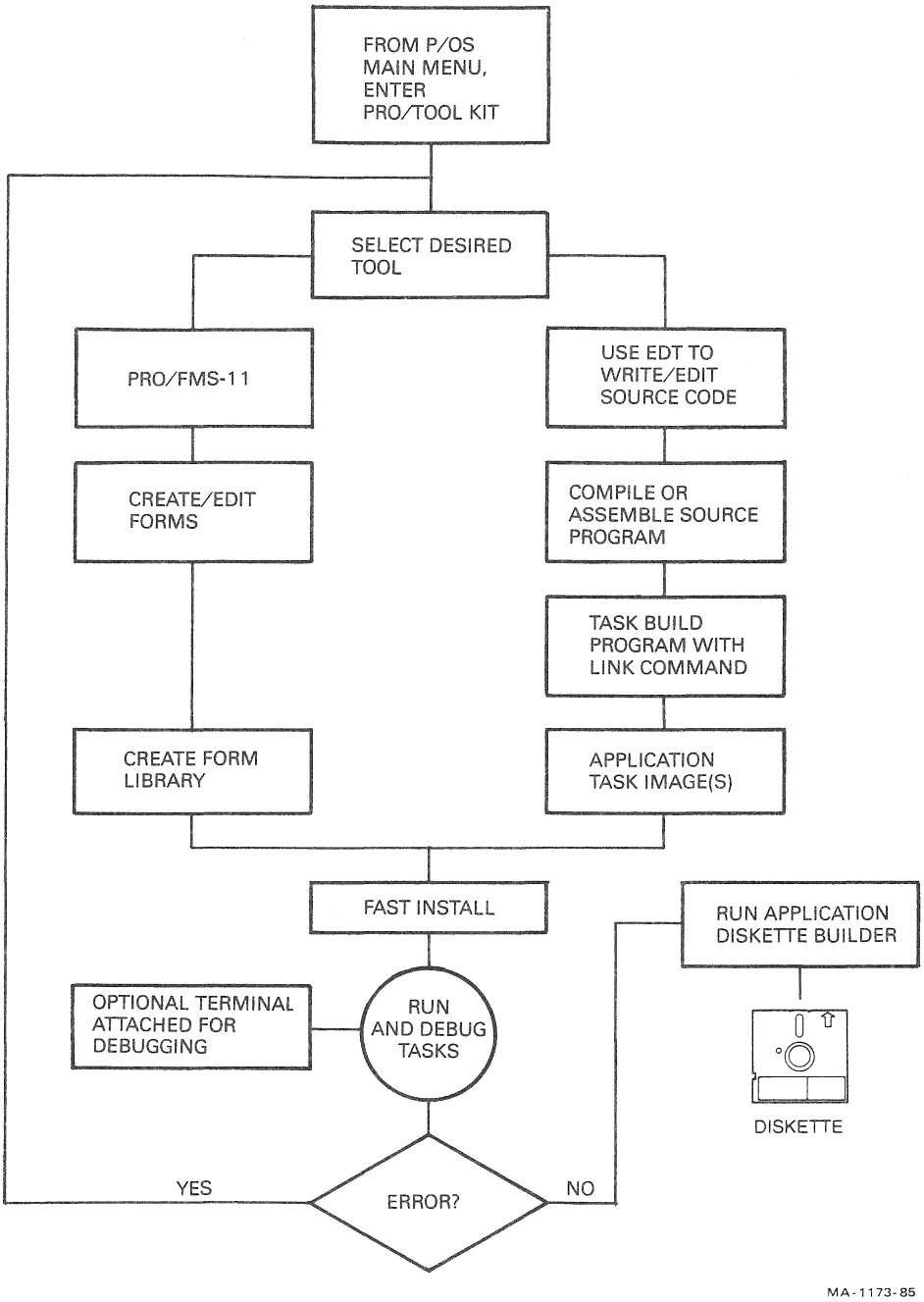

Figure 1-1: PRO/Tool Kit Development Cycle

PRO/FMS DEVELOPMENT CYCLE

Figure 1-2 illustrates the development cycle with the Host Tool Kit.

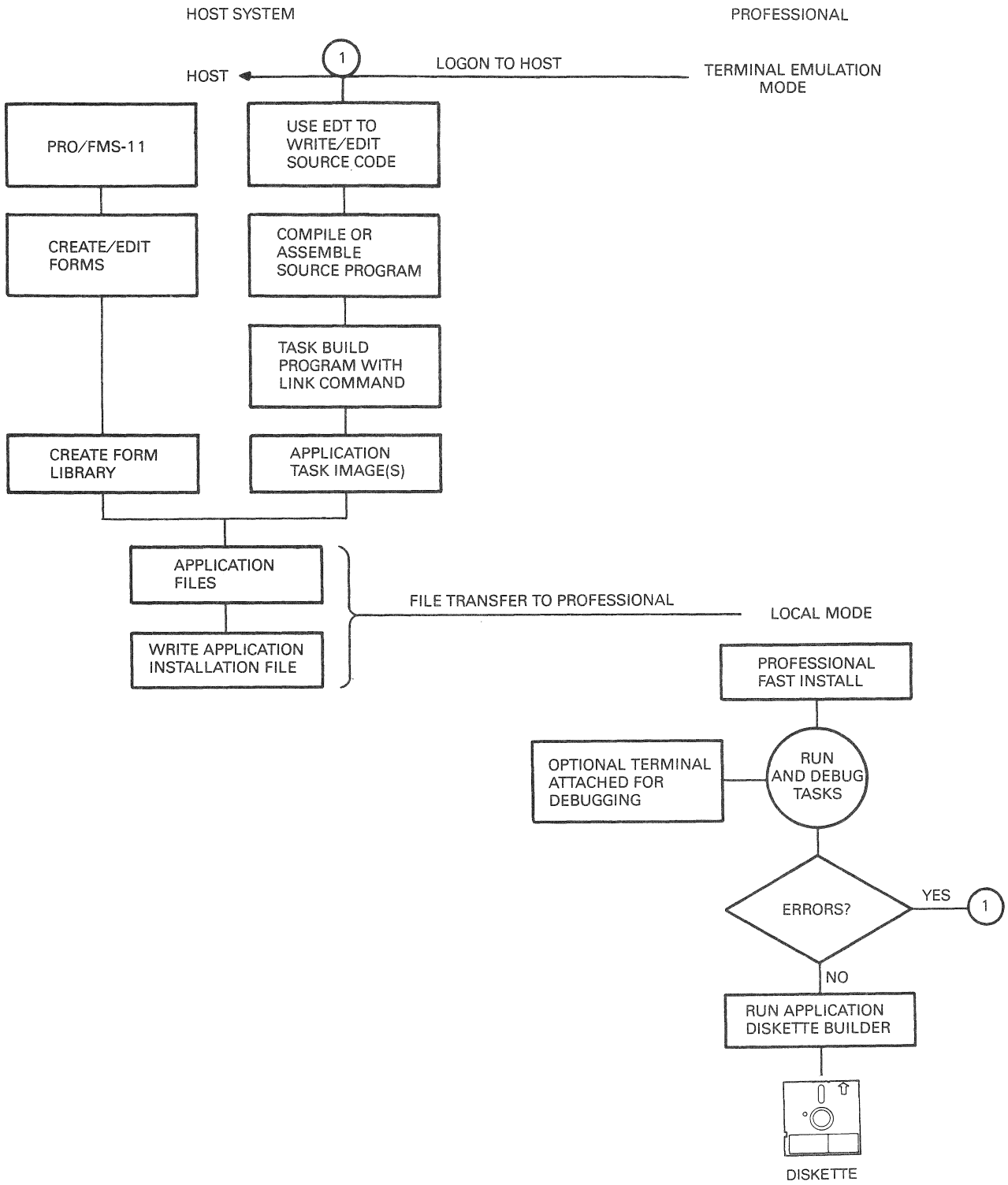

Figure 1-2: Host Tool Kit Development Cycle

MA-1174-85

CHAPTER 2

THE FORM EDITOR, PROFED

2.1 RUNNING PROFED IN TERMINAL EMULATION

To run PROFED in terminal emulation, set your terminal to VT200 mode.

To use DEC multinational 8-bit mode characters, you must also:

- Set the Professional Terminal Emulator to 8-bit mode and no parity by using the Line entry on the Communications Set-up Menu.

- Set the host to 8-bit mode.

  To do this on RSX (Host Tool Kit), type:

  ```
  > SET /EBC=TI:
  ```

  To do this on VMS (Host Tool Kit), type:

  ```
  $ SET TERMINAL/EIGHT_BIT
  ```

The default is 7-bit mode.

The following features are available once your system is in 8-bit mode:

- Field-marker characters include characters 241-277 octal.

- Background text includes the entire DEC Multinational Character Set.

- During application execution, PROFED forms (named data, form wide attributes, and field attributes) accept the DEC Multinational Character Set as input where any displayable character is requested.

The DEC Multinational Character Set uses 8-bit codes. You must

create the forms in 8-bit mode to use 8-bit codes in form descriptions. (See the *Terminal Subsystem Manual* for more information on the DEC Multinational Character Set.) If you create your forms in 7-bit mode, you can use only the ACSII character set, not the DEC Multinational Character Set.

2.2 ATTRIBUTE DIFFERENCES

The following FMS attributes operate differently on the Professional than on the VT100 running under FMS/RSX-11:

- **Bold field.** Not supported in wide-screen (132-column) mode.
- **Bold reverse video field.** Not recommended because of poor readability.
- **Blink field.** Can detract from PRO/FMS-11 performance if used excessively.

These differences may affect Professional system performance and/or form reliability. You should avoid these attributes when creating PRO/FMS-11 applications.

Also, note that PRO/FMS-11 allows fixed decimal fields to include a comma in place of a period for European applications.

2.3 THE PROFESSIONAL KEYBOARD AND THE FORM EDITOR

PRO/FMS uses different keys for some editing functions. Table 3-1 lists these differences.

CHAPTER 3

THE FORM DRIVER

The form library file specification must be a valid file specification. For more information on file specifications, refer to the *Tool Kit User's Guide*.

To determine the device and directory of the application, use the TRALOG system directive to translate the logical APPL$DIR. Refer to the *P/OS System Reference Manual* for more details.

The Form Driver can access a maximum of 60 forms per library. If you need more than 60 forms for your application, you must place the forms in multiple libraries (up to 4). You can access your forms in two ways:

1. When you need forms from another library, close the current library and open another library file. This option is slower than the second, but requires less data space.

2. You may have multiple impure areas with a library open in each. You can switch between impure areas to access the forms in each respective library. (See the sample BASIC program MULLIB in Chapter 7). This option is faster than the first, but requires more data space.

3.1 THE PROFESSIONAL KEYBOARD AND THE FORM DRIVER

With the Form Driver, PRO/FMS-11 uses different keys for some field terminators and interactive functions. Tables 3-1 and 3-2 list the differences.

Table 3-1: Keyboard Differences -- All Regions

FUNCTION	VT100	Professional
Enter form	ENTER/RETURN	DO, ENTER/RETURN
Move to next field	TAB	F12, TAB
Move to previous field	BACKSPACE	F11
Erase character	DELETE	<X
Erase field	LINEFEED	REMOVE
Insert/Overstrike	PF1	F13
Help	PF2	HELP
Repaint screen	CTRL/W	F20

Table 3-2: Keyboard Differences -- Scrolled Regions

FUNCTION	VT100	Professional
Exit field backward	PF1	F17
Exit field forward	PF2	F18

NOTE

On the Professional you can display a help form by using the <RESUME> key instead of the <DO> key, and the <NEXT SCREEN> key instead of the <HELP> key.

3.2 LINKING WITH THE FORM DRIVER LIBRARY

PRO/FMS-11 provides two versions of the Form Driver Library:

- Object Module
- Cluster Library

You must link your application with one of these libraries. Before you can link your application with either library, you must perform two preliminary steps.

1. Because the Form Driver requires P/OS User Interface Service Routines support, you must edit your command file as described in the section on "POSRES Task Image Requirements" in the *Tool Kit User's Guide*.

2. You must also edit your command file for the language you are using. See language documentation for details.

After performing these steps, you can link your program with the library you selected. In either case, you must edit the command (.CMD) and descriptor (.ODL) files to include the Form Driver library of your choice.

This chapter describes how to link your program with the object module. Chapter 4 describes how to link your program with the cluster library.

You can use the object module version of the Form Driver with P/OS V1.0 or later.

Suppose your source code is in BASIC-PLUS-2. The BASIC-PLUS-2 compiler generates a command file (filename.CMD) and an overlay descriptor language file (filename.ODL) when you enter the BUILD command. The following sections describe how to edit the .CMD and .ODL files.

3.2.1 Editing the .CMD File

After the preliminary editing described above, the .CMD file generated by the BASIC-PLUS-2 compiler for an application named "BASDEM" would look something like this:

```
SY:BASDEM/CP=SY:BASDEM/MP
TASK = BASDEM
UNITS = 19
ASG = TI:13:15
ASG = SY:1:5:6:7:8:9:10:11:12
EXTTSK= 952
CLSTR = PBESML,POSRES,RMSRES:RO
EXTSCT  = MN$BUF:0      ; STATIC SINGLE CHOICE MENU
EXTSCT  = DM$BUF:0      ; DYNAMIC SINGLE CHOICE MENU
EXTSCT  = HL$BUF:3410   ; HELP TEXT/MENU
EXTSCT  = MM$BUF:0      ; MULTI-CHOICE MENU
EXTSCT  = FL$BUF:0      ; FILE SELECTION/SPECIFICATION
GBLDEF  = MN$LUN:20     ; MENU FRAME FILE
GBLDEF  = HL$LUN:21     ; HELP FRAME FILE
GBLDEF  = MS$LUN:16     ; MESSAGE FRAME FILE
GBLDEF  = TT$LUN:15     ; TERMINAL I/O EVENT FLAG
GBLDEF  = MB$LUN:23     ; MESSAGE/STATUS DISPLAY
GBLDEF  = WC$LUN:22     ; DIRECTORY SEARCHES FOR OLDFIL and NEWFILE
GBLDEF  = TT$EFN:1      ; I/O EVENT FLAG
//
```

To use your command file with PRO/FMS-11, you must edit it. Because FDV uses Logical Unit 5 for input and output to the terminal, edit the command file by doing the following:

- Find the lines beginning with "ASG" and edit them to read:

```
ASG = TI:13:15:5
ASG = SY:1:6:7:8:9:10:11:12
```

- Set the extend section (EXTSCT) command to:

```
EXTSCT  = HL$BUF:3500   ; HELP TEXT/MENU
```

If your application uses the P/OS Help Services (described in Chapter 5), compute your frame size and set the extension to this value if it is larger than the current value. See the *Tool Kit User's Guide* for information on calculating frame size.

The fully edited command file for "BASDEM" would look like this (changed lines appear with an asterisk):

```
    SY:BASDEM/CP=SY:BASDEM/MP
    TASK = BASDEM
    UNITS = 19
*   ASG = TI:13:15:5
*   ASG = SY:1:6:7:8:9:10:11:12
    EXTTSK= 952
    CLSTR = PBESML,POSRES,RMSRES:RO
    EXTSCT  = MN$BUF:0      ; STATIC SINGLE CHOICE MENU
    EXTSCT  = DM$BUF:0      ; DYNAMIC SINGLE CHOICE MENU
*   EXTSCT  = HL$BUF:3500   ; HELP TEXT/MENU
    EXTSCT  = MM$BUF:0      ; MULTI-CHOICE MENU
    EXTSCT  = FL$BUF:0      ; FILE SELECTION/SPECIFICATION
    GBLDEF  = MN$LUN:20     ; MENU FRAME FILE
    GBLDEF  = HL$LUN:21     ; HELP FRAME FILE
    GBLDEF  = MS$LUN:16     ; MESSAGE FRAME FILE
    GBLDEF  = TT$LUN:15     ; TERMINAL I/O EVENT FLAG
    GBLDEF  = MB$LUN:23     ; MESSAGE/STATUS DISPLAY
    GBLDEF  = WC$LUN:22     ; DIRECTORY SEARCHES FOR OLDFIL and NEWFILE
    GBLDEF  = TT$EFN:1      ; I/O EVENT FLAG
    //
```

3.2.2 Editing the .ODL File for Media Resident Forms

Your application must reference the PRO/FMS-11 Form Driver, either the non-debug version (FDV) or the debug version (FDVDBG). To do this, edit the .ODL file to include the Form Driver as part of the root segment of the program, concatenated with the object module.

For example, to include the non-debug Form Driver in a BASIC-PLUS-2 .ODL file, change the line containing .ROOT from this:

```
.ROOT BASIC2-RMSROT-USER,RMSALL
```

to this:

```
.ROOT BASIC2-RMSROT-USER-FDV,RMSALL
```

In addition to including the Form Driver (either FDV or FDVDBG) as part of the line containing .ROOT, you must insert a new FDV: .FCTR line after the LIBR: line of your ODL file. Examples of the FDV: .FCTR lines for each Tool Kit language follow.

- **BASIC-PLUS-2**

  For the non-debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLBP2-LB:[1,5]FDV/LB
  ```

  For the debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLBP2-LB:[1,5]FDVDBG/LB
  ```

- **COBOL-81**

  Refer to Section 7.2 for more information and sample programs. In most cases, you should build with the HLLCOB (COBOL high level language FMS support) module. (This means that calls to FDV in your application pass string parameters without delimiters BY DESCRIPTOR.)

  If you use the non-debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLCOB-LB:[1,5]FDV.OLB/LB
  ```

  If you use the debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLCOB-LB:[1,5]FDVDBG.OLB/LB
  ```

  If, however, your COBOL program calls FDV only BY REFERENCE (the default), and, therefore, has delimiters on the string parameters (for example, for migrated PDP-11 COBOL V4.0 programs), you need to build with the HLLCBL module.

  If you use the non-debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLCBL-LB:[1,5]FDV.OLB/LB
  ```

  If you use the debug version of the Form Driver, add the line:

  ```
  FDV: .FCTR LB:[1,5]HLLCBL-LB:[1,5]FDVDBG.OLB/LB
  ```

- **FORTRAN-77**

  For the non-debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]HLLFOR-LB:[1,5]FDV/LB

  For the debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]HLLFOR-LB:[1,5]FDVDBG/LB

- **MACRO-11**

  For the non-debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]FDV/LB

  For the debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]FDVDBG/LB

- **PASCAL**

  PASCAL uses the FORTRAN version of the PRO/FMS-11 Form Driver. For the non-debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]HLLFOR-LB:[1,5]FDV/LB

  For the debug version of the Form Driver, add the line:

  FDV: .FCTR LB:[1,5]HLLFOR-LB:[1,5]FDVDBG/LB

3.2.3 Editing the .ODL File For Memory Resident Forms

If you use memory resident forms with the object module version of the Form Driver, you must add the object module that contains the form data (FORMS) to the .ODL file. For example, to use memory resident forms in a BASIC-PLUS-2 .ODL file, change the FDV: .FCTR line from this:

```
FDV: .FCTR LB:[1,5]HLLBP2 - LB:[1,5]FDV/LB
```

to this:

```
FDV: .FCTR LB:[1,5]HLLBP2 - LB:[1,5]FDV/LB - FORMS
```

where FORMS is the name of the object file created by the Forms Utility (PROFUT).

3.3 EXAMPLE FILES USING THE OBJECT MODULE LIBRARY

The following example shows an .ODL file before and after editing. The original .ODL file for a BASIC-PLUS-2 application named "BASDEM" would look like this:

```
                .ROOT BASIC2-RMSROT-USER,RMSALL
        USER:   .FCTR SY:BASDEM-LIBR
        LIBR:   .FCTR LB:[1,5]PBEOTS/LB
        @LB:[1,5]PBEIC1
        @LB:[1,5]RMSRLX
                .END
```

Edited to reference the non-debug version of the PRO/FMS-11 Form Driver, the BASIC-PLUS-2 .ODL file would look like this (new lines appear with an asterisk):

```
*               .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
        USER:   .FCTR SY:BASDEM-LIBR
        LIBR:   .FCTR LB:[1,5]PBEOTS/LB
*       FDV:    .FCTR LB:[1,5]HLLBP2-LB:[1,5]FDV/LB
        @LB:[1,5]PBEIC1
        @LB:[1,5]RMSRLX
                .END
```

The debug version of the edited BASIC-PLUS-2 .ODL would look like this:

```
*                .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
          USER:  .FCTR SY:BASDEM-LIBR
          LIBR:  .FCTR LB:[1,5]PBEOTS/LB
*         FDV:   .FCTR LB:[1,5]HLLBP2-LB:[1,5]FDVDBG/LB
          @LB:[1,5]PBEIC1
          @LB:[1,5]RMSRLX
                 .END
```

See the programs in Chapter 7 of this manual for sample edited .ODL files for other languages.

3.4 CHANGE IN FLOPEN

Before V2.0 of PRO/FMS-11, the Form Driver call to FLOPEN always returned a successful status. In V2.0 and later, FLOPEN returns the correct status. Refer to the *FMS-11/RSX Software Reference Manual* for the correct status values. Use the FSTAT call to check the file's status when issuing calls.

CHAPTER 4

FORM DRIVER CLUSTER LIBRARIES

Starting with V2.0 of the Tool Kit, the PRO/FMS-11 Form Driver is available in two cluster libraries:

- A non-debug version named FMSRES which comes with P/OS.
- A debug version named FMSDBG. This cluster library is shipped with the Tool Kit on the diskette labelled PRO/APP DSKT BLDR. See Chapter 6 for installation of this library.

The V1.7 object module Form Driver will continue as part of the Host Tool Kit for backward compatibilty. The new functions described in this chapter and the changes to the key function correlation exist *only* in the cluster libraries. Also, DIGITAL will enhance only the cluster libraries, providing support to V1.7 of the object module Form Driver.

Existing applications will run as they currently do. In order to take advantage of the new functions, however, you must link your application with the new cluster library. Source code changes are only necessary to take advantage of the new FNKON and FNKOFF functions.

4.1 NEW CALLS TO THE CLUSTER LIBRARY

The two new calls to the Form Driver (for higher level languages) are FNKON and FNKOFF. For MACRO programmers, the new function codes are FON (for FNKON) and FOF (for FNKOF).

4.1.1 FNKON - Turn On Function Key Processing

This call turns on function key processing. If you press a function key that PRO/FMS-11 doesn't use for its editing and form control functions, the terminator value for that function key is returned to the calling program.

BASIC-PLUS-2 and FORTRAN call:

```
CALL FNKON
```

COBOL call:

```
CALL "FNKON"
```

MACRO-11 call:

```
$FDV   ARG=arglst, FNC=FON, REQ=reqlst
```

Table 4-1 lists the returned status values and codes for the FNKON call.

Table 4-1: Returned Status Values and Codes for FNKON Call

Status Value High-Level Languages	Status Code Macro-11	Meaning
1	FS$SUC	Successful completion
-20	none	Wrong number of arguments in call (High-level languages only)
-21	none	Impure area not yet initialized (High-level languages only)

4.1.2 FNKOFF - Turn Off Function Key Processing

This call turns off function key processing. If the terminal operator presses one of the function keys that PRO/FMS-11 doesn't recognize, the terminal bell rings.

BASIC-PLUS-2 and FORTRAN call:

```
CALL FNKOFF
```

COBOL call:

```
CALL "FNKOFF"
```

MACRO-11 call:

```
$FDV   ARG=arglst, FNC=FOF, REQ=reqlst
```

Table 4-2 lists the returned status values and codes for the FNKOFF call.

Table 4-2: Returned Status Values and Codes for FNKOFF Call

Status Value High-Level Languages	Status Code Macro-11	Meaning
1	FS$SUC	Successful completion
-20	none	Wrong number of arguments in call (High-level languages only)
-21	none	Impure area not yet initialized (High-level languages only)

NOTE

The call to the Form Driver to process field terminators (for example, FPFT) returns the value of an undefined terminator if one of the special function key terminators is passed to it. These function key terminators have no meaning to PRO/FMS-11.

4.2 LK201 FUNCTION KEY TERMINATOR VALUES

The PRO/FMS-11 Cluster Libraries include support to return most of the LK201 function keys as terminators.

To use this new capability, you must change your source code and link with the cluster libraries. If you do not want to use this new capability, you can still build the application against the PRO/FMS-11 cluster libraries. The default for function key processing does *not* return the function keys as terminators.

If the current mode is set to pass back function keys, the form driver returns the function key to the application in the terminator variable. However, if the current mode is to disallow function keys, the form driver rings the terminal bell when the terminal operator presses a function key.

Table 4-3 lists the terminators for the LK201 function keys. All key values are in decimal.

Table 4-3: Function Key Terminator Values

Professional Label Strip	PRO/FMS-11 V2.0 High-Level Languages	Keycode Name Macro Global Symbols
E1 (Find)	33	FT$FND
E2 (Insert Here)	34	FT$INS
E3 (Remove)	35	FT$RMV
E4 (Select)	36	FT$SEL
E5 (Previous Screen)	37	FT$PRS
E6 (Next Screen)	38	FT$NXS
F1 (Hold Screen)	*	*
F2 (Print Screen)	*	*
F3 (Break)	46	FT$BRK
F4 (Set-Up)	47	FT$SET

LK201 FUNCTION KEY TERMINATOR VALUES

Professional Label Strip	PRO/FMS-11 V2.0 High-Level Languages	Keycode Name Macro Global Symbols
F5 (F5)	48	FT$F5K
F6 (Interrupt)	*	*
F7 (Resume)	50	FT$RSM
F8 (Cancel)	51	FT$CAN
F9 (Main Screen)	52	FT$MSC
F10 (Exit)	53	FT$EXI
F11 (Esc)	55	FT$F11
F12 (Bs)	2	FT$PRV
F13 (Lf)	*	*
F14 (Addtnl Options)	58	FT$AOP
Help	*	*
Do	0	FT$NTR
F17	63	FT$F17
F18	64	FT$F18
F19	65	FT$F19
F20	66	FT$F20

* Function values are not returnable to the application.

NOTE

Under certain circumstances the following two keys are not returned to the application. If the current displayed form is an FMS or FDT help form, the RESUME and NEXT SCREEN key are not returned. Instead, the RESUME key signals that the terminal operator has completed the help function. The NEXT SCREEN key displays the next HELP form, if any.

4.3 MAPPING FIELD TERMINATORS AND EDITING FUNCTIONS

The Cluster Libraries use different keys for the field terminators and editing functions than the object module version. These changes make PRO/FMS-11 more compatible with VAX-11 FMS.

Tables 4-4 and 4-5 show the differences between the cluster library and object module version of the PRO/FMS-11 Form Driver's mapping of function keys.

Table 4-4: Mapping Function Keys - All Regions

Function	Object Module Form Driver	Cluster Libraries Form Driver
Move to Previous Field	TAB, F11	TAB
Move to Next Field	F12	F12 (BS)
Erase Field	Remove	F13 (LF)
Insert	F13 *	PF1/PF3
Overstrike	F13 *	PF3
Help	HELP	PF2, HELP
Repaint Screen	F20	<CTRL/W>, <CTRL/R>

* Toggles current mode

Table 4-5: Mapping Function Keys - Scrolled Regions

Function	Object Module Form Driver	Cluster Libraries Form Driver
Exit Field Backward	F17	PF1/Up Arrow
Exit Field Forward	F18	PF1/Down Arrow

4.4 LINKING WITH THE CLUSTER LIBRARIES

Before you can link your program with the cluster library, you must perform the preliminary steps described in Section 3.2. Then you can edit your command (.CMD) and descriptor (.ODL) files to include the cluster library Form Driver.

The following sections describe how to edit the .CMD and .ODL files using the Basic-Plus-2 example "BASDEM" listed in Section 3.2.1.

4.4.1 Editing the .CMD File

Edit the "ASG" lines and extend section file (EXTSCT) as described in section 3.2.1.

Next, find the line:

```
CLSTR = PBESML,RMSRES,POSRES:RO
```

Modify it to include the PRO/FMS-11 cluster library (for the non-debug version):

```
CLSTR = PBESML,FMSRES,RMSRES,POSRES:RO
```

Or (for the debug version):

```
CLSTR = PBESML,FMSDBG,RMSRES,POSRES:RO
```

4.4.2 Editing the .ODL File for Media Resident Forms

Edit the .ROOT line to include either the non-debug (FDV) or debug (FDVDBG) version of the Form Driver, just as in Section 3.2.2.

However, the .FCTR line is different for the cluster library. After the LIBR: line, insert the following .FCTR line:

```
FDV:  .FCTR  LB:[1,5]HLLBP2
```

4.4.3 Editing the .ODL File for Memory Resident Forms

If you want to use memory resident forms with the cluster library, edit the .FCTR line as follows:

For the non-debug cluster library, change the line to read:

```
FDV:  .FCTR  LB:[1,5]HLLBP2 - LB:[1,5]SYSLIB/LB:FDVDAT - FORMS
```

where FORMS is the object module created by the Forms Utility containing your forms.

For the the debug cluster library, change the line to read:

```
FDV:  .FCTR  LB:[1,5]HLLBP2 - LB:[1,5]SYSLIB/LB:FDVDBG - FORMS
```

where FORMS is the object module created by the Forms Utility containing your forms.

4.4.4 Editing Your Installation (.INS) File

Add the appropriate line to your application installation (.INS) file.

If you built against the non-debug cluster library, add the following line:

```
INSTALL [ZZSYS]FMSRES.TSK/LIBRARY
```

If you built against the debug cluster library, add the following line:

```
INSTALL [ZZSYS]FMSDBG.TSK/LIBRARY
```

NOTE

Remember--you can build against a maximum of six cluster libraries.

4.5 EXAMPLE FILES USING THE CLUSTER LIBRARY

The following is the BASIC-PLUS-2 example .CMD file edited to link the application with the non-debug version of the PRO/FMS-11 cluster library. (Changed lines appear with an asterisk.)

```
    SY:BASDEM/CP/-FP,BASDEM/-SP=SY:BASDEM/MP
    TASK = BASDEM
    UNITS = 19
*   ASG = TI:13:15:5
*   ASG = SY:1:6:7:8:9:10:11:12
    EXTTSK= 952
*   CLSTR=PBESML,FMSRES,POSRES,RMSRES:RO
    EXTSCT=MN_$BUF:0     ;STATIC SINGLE CHOICE MENU
    EXTSCT=DM_$BUF:0     ;DYNAMIC SINGLE CHOICE MENU
*   EXTSCT=HL_$BUF:3500 ;HELP TEXT/MENU
    EXTSCT=MM_$BUF:0     ;MUTLI-CHOICE MENU
    EXTSCT=FL_$BUF:0     ;FILE SELECTION/SPECIFICATION
    GBLDEF=MN_$LUN:20    ;MENU FRAME FILE
    GBLDEF=HL_$LUN:21    ;HELP FRAME FILE
    GBLDEF=MS_$LUN:16    ;MESSAGE FRAME FILE
    GBLDEF=TT_$LUN:15    ;TERMINAL I/O EVENT FLAG
    GBLDEF=MB_$LUN:23    ;MESSAGE/STATUS DISPLAY
    GBLDEF=WC_$LUN:22    ;DIRECTORY SEARCHES FOR OLDFIL AND NEWFILE
    GBLDEF=TT_$EFN:1     ;I/O EVENT FLAG
    //
```

The following is the BASIC-PLUS-2 example .ODL file modified to link the application with the non-debug version of the PRO/FMS-11 cluster library.

```
*          .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
    USER:    .FCTR SY:BASDEM-LIBR
    LIBR:    .FCTR LB:[1,5]PBEOTS/LB
*   FDV:     .FCTR LB:[1,5]HLLBP2
    @LB:[1,5]PBEIC1
    @LB:[1,5]RMSRLX
             .END
```

CHAPTER 5

ENHANCEMENTS TO PRO/FMS-11 HELP FACILITIES

PRO/FMS-11 applications can use P/OS help frames in addition to regular FMS help forms.

When the terminal operator presses the HELP key while using an FMS form, help appears in the following order:

1. A one-line help message for the current field.

2. The help form specified by the current form.

3. The P/OS help frame specified in the Named Data section of the current form.

When the terminal operator presses the HELP key again or presses the NEXT SCREEN key, help appears in this order:

1. The help form specified by the current form.

2. The P/OS help frame specified in the Named Data section of the last form displayed.

Follow these steps to provide P/OS Help Services with PRO/FMS-11:

Step 1: **Create Help Frame Files**

Use the Frame Development Tool (FDT) to create help frame files. See the *Toolkit Reference Manual* for details.

Step 2: **Run PRO/FMS-11 Forms Editor**

Run the PRO/FMS-11 Forms Editor (PROFED) and retrieve the form. Enter the NAME command. PROFED will display the Named Data Entry Form.

Step 3: **Enter .HELP.**

In Named Data Entry Form, enter .HELP. in the Name field; and the frame identifier (frameid) in the Data field.

Step 4: **Edit the Source Code**

Edit the source code so that it includes calls to P/OS Help Services to open the Help file before it calls the Form Driver, and to close the HELP file before the program exits.

Step 5: **Edit Command File**

Make sure you edit the command file according to Section 3.2.1.

Step 6: **Run PRO/FMS-11 Forms Utility**

Run the PRO/FMS-11 Forms Utility (PROFUT) and replace the old form with the new one.

A completed Named Data Entry form would look something like this:

```
 Name      Data
!----! !-------------------------------------------------!
.HELP.    INFO2
```

CHAPTER 6

INSTALLING OPTIONAL APPLICATIONS

Several optional applications are on the diskette labelled PRO/APP DSKT BLDR, supplied with both the Host Tool Kit and the PRO/Tool Kit. Install and remove them as needed.

To use the Debug Form Driver, copy the following files:

- FMSDBG.TSK
- FMSDBG.MSG

To run the example programs listed in Chapter 7, copy the following files:

- DEMLIB.FLB
- LIBRARY1.FLB
- LIBRARY2.FLB

Copy these files to your Professional from the diskette labelled PRO/APP DSKT BLDR as follows.

1. Log into a system manager's account or a privileged account.

2. Insert the diskette into a diskette drive slot.

3. Copy the files as follows, using either PRO DCL or P/OS File Services. All files are in directory [PROFMS] on the volume labelled TOOLKIT.

```
Copy TOOLKIT:[PROFMS]FMSDBG.TSK to LB:[ZZSYS]
Copy TOOLKIT:[PROFMS]FMSDBG.MSG to LB:[001002]
Copy TOOLKIT:[PROFMS]DEMLIB.FLB to LB:[001002]
Copy TOOLKIT:[PROFMS]LIBRARY1.FLB to LB:[001002]
Copy TOOLKIT:[PROFMS]LIBRARY2.FLB to LB:[001002]
```

The optional files are now on your system.

CHAPTER 7

SAMPLE PRO/FMS-11 PROGRAMS

Sample PRO/FMS-11 programs for each of the Tool Kit languages are part of the Tool Kit in directory LB:[1,5].

Copy the files to your own area. The accompanying forms are in the file DEMLIB.FLB on the PRO/APP DSKT BLDR diskette distributed with both the Host Tool Kit and the PRO/Tool Kit. Use the instructions in Chapter 6 to copy the file to your Professional.

This section contains listings of the sample programs, with any restrictions or comments you may need.

7.1 TOOL KIT BASIC-PLUS-2

BASIC-PLUS-2 programs can use either the Object Module Form Driver or Cluster Library Form Driver.

7.1.1 BASDEM

The first BASIC-PLUS-2 program, BASDEM, is an FMS-11/RSX generic program modified to run on P/OS. It demonstrates use of an Object Module Form Driver.

```
100 REM
110 REM BASDEM.B2S
120 REM
130 REM
140 REM                                   COPYRIGHT (C) 1979 BY
150 REM                   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
160 REM
170 REM
180 REM MODULE:           BASDEM
190 REM
200 REM VERSION: V01.00
210 REM
220 REM AUTHOR:           MEGAN
230 REM
240 REM DATE:             10-APRIL-79
245 REM
250 REM MODIFIED:
255 REM
257 REM DATE:             11-MARCH-1983   Changed Command and Odl file
260 REM
270 REM BASIC-PLUS-2 V2.0 demonstration program for FMS illusrating a
280 REM simple form-driven, data entry application.
290 REM
292 REM Below is an example of a command and ODL file to build
294 REM this demonstration program.
296 REM
298 REM ;
300 REM ;         BASDEM.CMD
301 REM SY:BASDEM/CP/FP,BASDEM/-SP=SY:BASDEM/MP
302 REM TASK = BASDEM
303 REM UNITS = 19
304 REM ASG = TI:13:15:5
305 REM ASG = SY:1:6:7:8:9:10:11:12
306 REM EXTTSK = 952
307 REM CLSTR=PBESML,POSSUM,POSRES,RMSRES:RO
308 REM EXTSCT=MN$BUF:0          ;SINGLE CHOICE MENU
309 REM EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE
310 REM EXTSCT=HL$BUF:3410       ;HELP
312 REM EXTSCT=MM$BUF:0          ;MULTI-CHOICE MENU
313 REM EXTSCT=FL$BUF:0          ;MULTI-CHOICE MENU
314 REM GBLDEF=MN$LUN:22         ;MENU
315 REM GBLDEF=HL$LUN:20         ;HELP
316 REM GBLDEF=MS$LUN:21         ;MESSAGE
317 REM GBLDEF=TT$LUN:15         ;TERMINAL I/O LUN
318 REM GBLDEF=WC$LUN:23         ;FILE LUN
319 REM GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
320 REM //
321 REM
421 REM ;
422 REM ;         TKB overlay descriptor language file to build BASDEM
423 REM ;
424 REM           .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
425 REM USER:     .FCTR SY:BASDEM-LIBR
426 REM LIBR:     .FCTR LB:[1,5]PBEOTS/LB
427 REM FDV:      .FCTR LB:[1,5]HLLBP2-LB:[1,5]FDV/LB
428 REM @LB:[1,5]PBEIC1
429 REM @LB:[1,5]RMSRLX
430 REM           .END
440 REM
450 REM
500 REM           Defined variables.
501 DIM I%(1500)
502 C$=STRING$(2%,65%)
503 F1$=STRING$(6%,32%)
504 F2$=STRING$(13%,32%)
505 A$=STRING$(255%,32%)
```

```
510 REM VARIABLE                        DESCRIPTION
520 REM
530 REM C$                          Choice specified by the user
550 REM S%                          FDV status
560 REM T%                          Terminator code
570 REM F1$                         The initial form name of the series
580 REM F2$                         The output file name
590 REM F3$                         The current form name
600 REM
610 REM Initialize Form Driver and open library.
620 REM
625 CALL WTQIO(768%,5%,5%)
630 CALL FINIT(I%(),1500%)
635 CALL FLCHAN(6%) \ GOSUB 2000
640 CALL FLOPEN ("LB:[1,2]DEMLIB") \ GOSUB 2000
650 REM
660 REM Show the menu form for operator to select the data
670 REM collection series.  Get the first form name from
680 REM named data.
690 REM
700 CALL FCLRSH("FIRST") \ GOSUB 2000
710 CALL FGET(C$,T%,"CHOICE") \ GOSUB 2000
720 CALL FNDATA(C$,F1$) \ CALL FSTAT(S%) \ IF S%>0% GO TO 770
730 CALL FPUTL("Illegal choice") \ GO TO 710
740 REM
750 REM If form name is ".EXIT.", terminal operator is done.
760 REM
770 IF F1$=".EXIT." GO TO 1290
780 REM
790 REM Get the output file name from named data and open it.
800 REM
810 CALL FNDATA(TRM$(C$)+"F",F2$)
820 OPEN F2$ FOR OUTPUT AS FILE #1%, FILESIZE 10%
830 REM
840 REM         THIS IS THE DATA COLLECTION LOOP
850 REM
860 REM Set current form = first form in series.
870 REM
880 F3$=F1$
890 REM
900 REM Show the form.
910 REM
920 CALL FCLRSH(F3$) \ GOSUB 2000
930 REM
940 REM Get data for current form and output it.
950 REM
960 CALL FGETAL(A$) \ GOSUB 2000
970 PRINT #1%,TRM$(A$)
980 REM
990 REM Get name of next form.  If found, loop for more data.
1000 REM
1010 CALL FNDATA("NXTFRM",F3$)
1020 IF F3$<>".NONE."GO TO 920
1030 REM
1040 REM End of the form series.  Show last to determine if
1050 REM we're done or not.
1060 REM
1070 CALL FCLRSH("LAST") \ GOSUB 2000
1080 CALL FGET(C$,T%,"CHOICE") \ GOSUB 2000
1090 REM
1100 REM If response = "1", repeat data collection loop.
1110 REM
1120 IF C$="1" GO TO 880
1130 REM
1140 REM Get named data corresponding to response.
1150 REM Get field again if illegal response.
1160 REM Close output file for valid response other than 1.
1170 REM
```

```
1180 CALL FNDATA(C$,F3$) \ CALL FSTAT(S%) \ IF S%>0% GO TO 1200
1190 CALL FPUTL(Illegal choice") \ GO TO 1080
1200 CLOSE #1%
1210 REM
1220 REM If named data is ".EXIT.", terminal operator is done, else
1230 REM display menu form again.
1240 REM
1250 IF F3$<>".EXIT." GO TO 700
1260 REM Close form library and exit.
1270 REM
1280 REM
1290 CALL FLCLOS \ GO TO 9999
2000 REM
2010 REM Output message and exit if I/O error returned from
2020 REM form Driver.  This is the only error expected in a
2030 REM debugged application.
2040 REM
2050 CALL FSTAT(S%)
2060 IF S%>0% THEN RETURN
2070 CALL FPUTL("Fatal I/O Error") \ STOP
9999 END
```

7.1.2 MULLIB

The following BASIC-PLUS-2 program, MULLIB, demonstrates use of the Cluster Library Form Driver.

```
100 REM
110 REM MULLIB.B2S
120 REM
130 REM
140 REM                          COPYRIGHT (C>)1984 BY
150 REM          DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
160 REM
170 REM
180 REM MODULE:          MULLIB
190 REM
200 REM VERSION: V01.00
210 REM
220 REM AUTHOR:
230 REM
257 REM DATE:            8/4/84
260 REM
270 REM  BASIC- Plus- 2 V2.0 demonstration program for FMS illustrating
280 REM  access to multiple libraries at the same time.
290 REM
292 REM Below is an example of a command and ODL file to build
294 REM this demonstration program.
296 REM
298 REM ;
300 REM ;        MULLIB.CMD
301 REM SY:MULLIB/CP/-FP,MULLIB/-SP=SY:MULLIB/MP
302 REM TASK = MULLIB
303 REM UNITS = 19
304 REM ASG = TI:13:15:5
305 REM ASG = SY:1:6:7:8:9:10:11:12
306 REM EXTTSK= 952
307 REM CLSTR=PBESML,FMSRES,POSRES,RMSRES:RO
308 REM EXTSCT=MN$BUF:0          ;STATIC SINGLE CHOICE MENU
309 REM EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE MENU
310 REM EXTSCT=HL$BUF:3500       ;HELP TEXT/MENU
311 REM EXTSCT=MM$BUF:0          ;MUTLI-CHOICE MENU
312 REM EXTSCT=FL$BUF:0          ;FILE SELECTION/SPECIFICATION
```

```
313 REM GBLDEF=MN$LUN:1          ;MENU FRAME FILE
314 REM GBLDEF=HL$LUN:0          ;HELP FRAME FILE
315 REM GBLDEF=MS$LUN:10         ;MESSAGE FRAME FILE
316 REM GBLDEF=TT$LUN:15         ;TERMINAL I/O
317 REM GBLDEF=MB$LUN:11         ;MESSAGE/STATUS DISPLAY
318 REM GBLDEF=WC$LUN:12         ;DIRECTORY SEARCHES FOR OLDFIL AND NEWFILE
319 REM GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
320 REM //
321 REM
421 REM ;
422 REM ;        TKB command file to build MULLIB
423 REM ;
424 REM          .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
425 REM USER:    .FCTR SY:MULLIB-LIBR
426 REM LIBR:    .FCTR LB:[1,5]PBEOTS/LB
427 REM FDV:     .FCTR LB:[1,5]HLLBP2
428 REM @LB:[1,5]PBEIC1
429 REM @LB:[1,5]RMSRLX
430 REM          .END
440 REM
450 REM
500 REM          Defined Variables
510 REM
520 DIM I1%(1000)
530 DIM I2%(1000)
540 C$=STRING$(2%,32%)
550 A$=STRING$(100%,43%)
599 REM
600 REM VARIABLE                 DESCRIPTION
610 REM
620 REM I1%              First Impure Area
630 REM I2%              Second Impure Area
640 REM C$               Choice specified by the user
650 REM A$               Data returned from the form
699 REM
700 REM Initialize Form Driver and open first library
710 REM
720 CALL WTQIO (768%,5%,5%)
730 CALL FINIT (I1%(),2000%)
740 CALL FLCHAN (6%) \ GOSUB 2000
750 CALL FLOPEN ("LB:[1,2]LIBRARY1") \ GOSUB 2000
760 REM
770 REM Show the first form from the first library
780 REM
790 CALL FCLRSH ("FIRST") \ GOSUB 2000
800 CALL FGETAL (A$) \ GOSUB 2000
810 REM
820 REM Now open the second library using channel 7
830 REM
840 CALL FINIT (I2%(),2000%)
850 CALL FLCHAN (7%) \ GOSUB 2000
860 CALL FLOPEN ("LB:[1,2]LIBRARY2") \ GOSUB 2000
870 REM
880 REM Show the first form from the second library
890 REM
900 CALL FCLRSH ("FIRST") \GOSUB 2000
910 CALL FGETAL (A$) \ GOSUB 2000
1000 REM
1010 REM Now set the Impure back to the first library and
1020 REM display another form.
1020 REM
1030 CALL FINIT (I1%(),1000)
1040 CALL FLCHAN (6%) \ GOSUB 2000
1050 CALL FCLRSH ("SECOND") \ GOSUB 2000
1060 CALL FGETAL (A$) \ GOSUB 2000
1070 REM
1080 REM Close the first library
1090 REM
```

```
1100 CALL FLCLOS
1110 REM
1120 REM Now close the second library
1130 REM
1140 CALL FINIT (I2%(),1000)
1150 CALL FLCHAN (7%)
1160 CALL FLCLOS \ GOTO 9999
1170 REM
2000 REM Output message and exit if I/O error returned from
2010 REM the Form Driver. This is the only error expected in a
2020 REM debugged application.
2040 REM
2050 CALL FSTAT(S%)
2060 IF S%>0% THEN RETURN
2070 CALL FPUTL("Fatal I/O Error") \ STOP
9999 END
```

7.2 TOOL KIT COBOL-81

When calling PRO/FMS-11 from a COBOL-81 program,

- Use the By Descriptor argument-passing mechanism for character-type parameters.

- Use the By Reference argument-passing mechanism for numeric-type parameters, such as the starting line parameter of FCLRSH or the LUN for the FLCHAN call.

The ODL file should then use the HLLCOB interface. See Section 3.2.2 on editing the descriptor file.

Another option is available, however, which allows all parameter passing By Reference, but requires string delimiters on all character-type parameters. The following sample programs demonstrate each method.

7.2.1 Passing Variables by Descriptor

This COBOL program passes data variables by descriptor to the Form Driver. (Numeric variables are passed by reference.) It uses the interface HLLCOB.

```
*
*       CBLDESDEM.CBL
*
*                   COPYRIGHT (C) 1979 BY
*       DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
*
*       MODULE:         CBLDESDEM
*
*       VERSION:        V01.00
*
*       AUTHOR:
*
*       DATE:           1-APRIL-79
*
*       MODIFIED:       To run on the Professional
*
*       DATE:           3-MARCH-83
*
*       COBOL demonstration program for FMS illustrating a
*       simple form-driven, data entry application. This program
*       demonstrates the Call By Descriptor method to call the
*       Form Driver.
*
*       The following is a brief description on compiling and
*       building CBLDESDEM.
*
*       The command to compile the program is:
*
*               MCR PROC81 CBLDESDEM,CBLDESDEM=CBLDESDEM
*
```

```
*       Below is an example of a TKB command file to build
*       this demonstration program.
*
*       ;
*       ;       CBLDESDEM.CMD
*       ;
*       ;       TKB command file to build CBLDESDEM
*       ;
*       ;TKB COMMAND FILE CREATED ON 01-MAR-83 AT 14:16:02
*       CBLDESDEM/CP/-FP,CBLDESDEM/-SP=CBLDESDEM/MP
*       TASK=CBLDEM
*       CLSTR=C81LIB,POSRES,RMSRES:RO
*       ;
*       EXTSCT=MN$BUF:0          ;SINGLE CHOICE MENU
*       EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE
*       EXTSCT=HL$BUF:3410       ;HELP
*       EXTSCT=MS$BUF:3100       ;MESSAGE
*       EXTSCT=MM$BUF:0          ;MUTLI-CHOICE MENU
*       EXTSCT=FL$BUF:0          ;FILE SELECTION FOR OLDFIL
                                 ;AND NEWFIL
*       GBLDEF=MN$LUN:22         ;MENU
*       GBLDEF=HL$LUN:20         ;HELP
*       GBLDEF=TT$LUN:15         ;TERMINAL I/O
*       GBLDEF=WC$LUN:23
*       GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
*       ;
*       UNITS = 19
*       ASG = TI:13:15:5
*       ASG = SY:6:7:8:9:10:11:12
*       //
*
*  ------------------------------------------------------------------------


*
*       Below is an example ODL file to build the Demonstration Program
*
*       ;MERGED ODL FILE CREATED ON  01-MAR-83 AT 14:16:02
*       @CBLDESDEM.SKL
*       SCOBJ$: .FCTR CBLDESDEM.OBJ
*       @LB:[1,1]RMSRLX.ODL
*               .NAME RMS$TR
*       RMSTR$: .FCTR RMS$TR-RMSALL
*       RMS$:   .FCTR  RMSROT
*       SCLIB$: .FCTR LB:[1,1]C81LIB/LB
*       OBJRT$: .FCTR SCOBJ$-FDV-SCLIB$-RMS$
*       FDV:    .FCTR LB:[1,5]HLLCOB-LB:[1,5]FDV.OLB/LB
*               .ROOT OBJRT$,RMSTR$
*        .END
*
*


 IDENTIFICATION DIVISION.
 PROGRAM-ID.    CBLDEM.
*
*
*
*       TEST PROGRAM
*
*
 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.
*
D       OUTPUT-FILE
        LABEL RECORDS ARE STANDARD
        VALUE OF ID IS ANSWER2.
01      POOL PIC X(256).


*
*       Data follows.
*
 WORKING-STORAGE SECTION.
*       System form library.
01 DEMLIB       PIC X(25) VALUE "LB:[1,2]DEMLIB.FLB".
*       Logical unit number for FMS library file.
01 LUN          PIC 99 COMP VALUE 6.
*       Impure area.
01 IMPURE       PIC X(2000).
*       Size of impure area.
01 ISIZE        PIC 9999 COMP VALUE 2000.
01 INUM         PIC 999 COMP VALUE 768.
01 UN           PIC 9 COMP VALUE 5.
*
*       Special work area.
*       ANSWER1 -> The initial form name of the series.
*       ANSWER2 -> The output file name.
*       ANSWER3 -> The current form name.
*
01 ANSWER1.
  02 PART PIC X(6).
  02 FILLER PIC X(7).
01 ANSWER2      PIC X(13).
01 ANSWER3.
  02 PART PIC X(6).
  02 FILLER PIC X(7).
*
*       Fieldf used to create a field name.
*
01 FIELDF.
  02 DAT PIC X.
  02 FILLER PIC X(5) VALUE "F    ".
01 FIELD        PIC X(6).


*       Status
*
01 STAT PIC 99 COMP.
01 STAT2 PIC 99 COMP.
*
*       Error message on program errors.
*
01 ERR1.
  02 PART1      PIC X(23) VALUE "FATAL I/O ERROR, STAT=".
  02 ERR-STAT   PIC ZZZZ9- DISPLAY.
  02 PART2      PIC X(8) VALUE ", STAT2=".
  02 ERR-STAT2  PIC ZZZZ9- DISPLAY.
01 ILL-CHOICE   PIC X(16) VALUE "ILLEGAL CHOICE".


*
*
*       FORM DESCRIPTION STARTS HERE
*
*
COPY "LB:[1,5]DEMLIB.LIB".
*
*
*
*
```

```
PROCEDURE DIVISION.
MAIN-CONTROL SECTION.
P1.
*               Attach the terminal.
        CALL "WTQIO" USING INUM, UN, UN.
*               Initialize and open the library.
        CALL "FINIT" USING BY DESCRIPTOR IMPURE,
                           BY REFERENCE ISIZE.
        CALL "FLCHAN" USING BY REFERENCE LUN.
        PERFORM STATUS-CHECK.
        CALL "FLOPEN" USING BY DESCRIPTOR DEMLIB.
        PERFORM STATUS-CHECK.
*               Display first form.
P6.
        CALL "FCLRSH" USING BY DESCRIPTOR FORM-FIRST.
        PERFORM STATUS-CHECK.


*               Show the menu form for operator to select the data
*               collection series.  Get the first form name from
*               named data.
P2.
        CALL "FGET" USING BY DESCRIPTOR  D-FIRST-CHOICE,
                          BY REFERENCE   STAT,
                          BY DESCRIPTOR  N-FIRST-CHOICE.
        PERFORM STATUS-CHECK.
        MOVE  D-FIRST-CHOICE TO FIELD.
        MOVE SPACES TO ANSWER1.
        CALL "FNDATA" USING BY DESCRIPTOR  FIELD, ANSWER1.
        CALL "FSTAT" USING BY REFERENCE STAT.
        IF STAT NOT > 0
         CALL "FPUTL" USING BY DESCRIPTOR ILL-CHOICE
         PERFORM STATUS-CHECK
         GO TO P2.
*
*               If form name is ".EXIT.", terminal operator is done.
*
        IF PART OF ANSWER1 = ".EXIT." GO TO LIB-CLOSE.
*
*               Get the output file name from named data and open it.
*
        MOVE D-FIRST-CHOICE TO DAT OF FIELDF.
        MOVE SPACES TO ANSWER2.
        CALL "FNDATA" USING BY DESCRIPTOR
                       FIELDF, ANSWER2.
        PERFORM STATUS-CHECK.
        OPEN OUTPUT OUTPUT-FILE.
*
*               This is the data collection loop.
*
P4.
        MOVE ANSWER1 TO ANSWER3.
*               Show the form.
P3.
        CALL "FCLRSH" USING BY DESCRIPTOR ANSWER3.
        PERFORM STATUS-CHECK.


*
*               Get data for current form and output it.
*
        MOVE SPACES TO POOL.
        CALL "FGETAL" USING BY DESCRIPTOR POOL.
        PERFORM STATUS-CHECK.
        WRITE POOL.
```

```
*
*               Get name of next form.  If found, loop for more data.
*
        MOVE "NXTFRM" TO FIELD.
        CALL "FNDATA" USING BY DESCRIPTOR FIELD, ANSWER3.
        PERFORM STATUS-CHECK.
        IF PART OF ANSWER3 NOT = ".NONE." GO TO P3.
*
*               End of the form series.  Show last to determine if
*               we're done or not.
*
        CALL "FCLRSH" USING BY DESCRIPTOR FORM-LAST.
        PERFORM STATUS-CHECK.


P5.
        CALL "FGET" USING BY DESCRIPTOR D-LAST-CHOICE,
                          BY REFERENCE  STAT,
                          BY DESCRIPTOR N-LAST-CHOICE.
        PERFORM STATUS-CHECK.
        MOVE D-LAST-CHOICE TO FIELD.
*
*               If response = "1", repeat data collection loop.
*
        IF FIELD = "1" GO TO P4.
*
*               Get named data corresponding to response.
*               Get field again if illegal response.
*               Close output file for valid response other than 1.
*
        CALL "FNDATA" USING BY DESCRIPTOR FIELD, ANSWER3.
        CALL "FSTAT" USING BY REFERENCE STAT.
        IF STAT NOT > 0
         CALL "FPUTL" USING BY DESCRIPTOR ILL-CHOICE
         PERFORM STATUS-CHECK
         GO TO P5.
        CLOSE OUTPUT-FILE.
*
*               If named data is ".EXIT.", terminal operator
*               is done, else display menu form again.
*
        IF PART OF ANSWER3 NOT = ".EXIT." GO TO P6.
*
*               Close form library and exit.
*
LIB-CLOSE.
        CALL "FLCLOS".
        PERFORM STATUS-CHECK.
        STOP RUN.


*
*               Output message and exit if I.O error returned from
*               Form Driver.  This is the only error expected in a
*               debugged application.
*
 STATUS-CHECK SECTION.
 SC1.
        CALL "FSTAT" USING BY REFERENCE STAT, STAT2.
        IF STAT > 0 GO TO SC2.
        MOVE STAT TO ERR-STAT.
        MOVE STAT2 TO ERR-STAT2.
        DISPLAY ERR1 AT LINE 1 AT COLUMN 1, ERASE TO END OF SCREEN.
        DISPLAY "Press RESUME to continue." AT LINE 3 AT COLUMN 1.
        CALL "WTRES".
        STOP RUN.
 SC2.
        EXIT.
```

7.2.2 Passing Variables by Reference

This COBOL program passes all variables by reference to the Form Driver. It uses the interface HLLCBL.

```
*
*       CBLDEM.CBL
*
*                   COPYRIGHT (C) 1979 BY
*       DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
*
*       MODULE:          CBLDEM
*
*       VERSION:         V01.00
*
*       AUTHOR:
*
*       DATE:            1-APRIL-79
*
*       MODIFIED:        To run on the Professional
*
*       DATE:            3-MARCH-83
*
*       COBOL demonstration program for FMS illustrating a
*       simple form-driven, data entry application. This COBOL
*       example program uses the By Reference method (default)
*       for all calls to the Form Driver.
*
*       The following is a brief description on compiling and
*       building CBLDEM.
*
*       The command to compile the program is:
*
*               MCR PROC81 CBLDEM,CBLDEM=CBLDEM
*
*       Below is an example of a TKB command file to build
*       this demonstration program.
*
*       ;
*       ;       CBLDEM.CMD
*       ;
*       ;       TKB command file to build CBLDEM
*       ;
*       ;TKB COMMAND FILE CREATED ON 01-MAR-83 AT 14:16:02
*       CBLDEM/CP/-FP,CBLDEM/-SP=CBLDEM/MP
*       TASK=CBLDEM
*       CLSTR=C81LIB,POSRES,RMSRES:RO
*       ;
*       EXTSCT=MN$BUF:0          ;SINGLE CHOICE MENU
*       EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE
*       EXTSCT=HL$BUF:3410       ;HELP
*       EXTSCT=MM$BUF:0          ;MUTLI-CHOICE MENU
*       EXTSCT=FL$BUF:0          ;MULTI-CHOICE MENU
*       GBLDEF=MN$LUN:22         ;MENU
*       GBLDEF=HL$LUN:20         ;HELP
*       GBLDEF=MS$LUN:21         ;MESSAGE
*       GBLDEF=TT$LUN:15         ;TERMINAL I/O
*       GBLDEF=WC$LUN:23
*       GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
*       ;
*       UNITS = 19
*       ASG = TI:13:15:5
*       ASG = SY:6:7:8:9:10:11:12
*       //
*
```

```
*  ------------------------------------------------------------------------
*
*       Below is an example ODL file to build the Demonstration Program
*
*       ;MERGED ODL FILE CREATED ON  01-MAR-83 AT 14:16:02
*       @CBLDEM.SKL
*       SCOBJ$: .FCTR CBLDEM.OBJ
*       @LB:[1,1]RMSRLX.ODL
*               .NAME RMS$TR
*       RMSTR$: .FCTR RMS$TR-RMSALL
*       RMS$:   .FCTR  RMSROT
*       SCLIB$: .FCTR LB:[1,1]C81LIB/LB
*       OBJRT$: .FCTR SCOBJ$-FDV-SCLIB$-RMS$
*       FDV:    .FCTR LB:[1,5]HLLCBL-LB:[1,5]FDV.OLB/LB
*               .ROOT OBJRT$,RMSTR$
*        .END
*
*

 IDENTIFICATION DIVISION.
 PROGRAM-ID.    CBLDEM.
*
*
*
*       TEST PROGRAM
*
*
 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 ANSWER2.
01      POOL PIC X(256).


*
*       Data follows.
*
 WORKING-STORAGE SECTION.
*       System form library.
01 DEMLIB       PIC X(25) VALUE "#LB:[1,2]DEMLIB.FLB#".
*       Logical unit number for FMS library file.
01 LUN          PIC 99 COMP VALUE 6.
*       Impure area.
01 IMPURE       PIC X(2000).
*       Size of impure area.
01 ISIZE        PIC 9999 COMP VALUE 2000.
01 INUM         PIC 999 COMP VALUE 768.
01 UN           PIC 9 COMP VALUE 5.
*
*       Special work area.
*       ANSWER1 -> The initial form name of the series.
*       ANSWER2 -> The output file name.
*       ANSWER3 -> The current form name.
*
01 ANSWER1.
  02 PART PIC X(6).
  02 FILLER PIC X(7).
01 ANSWER2      PIC X(13).
01 ANSWER3.
  02 PART PIC X(6).
  02 FILLER PIC X(7).
```

```
*
*       Fieldf used to create a field name.
*
01 FIELDF.
  02 DAT PIC X.
  02 FILLER PIC X(5) VALUE "F    ".
01 FIELD        PIC X(6).


*       Status
*
01 STAT PIC 99 COMP.
01 STAT2 PIC 99 COMP.
*
*       Error message on program errors.
*
01 ERR1.
  02 PART1      PIC X(22) VALUE "FATAL I/O ERROR, STAT=".
  02 ERR-STAT   PIC ZZZZ9- DISPLAY.
  02 PART2      PIC X(8) VALUE ", STAT2=".
  02 ERR-STAT2  PIC ZZZZ9- DISPLAY.
01 ILL-CHOICE   PIC X(16) VALUE "#ILLEGAL CHOICE#".
*
*
*       FORM DESCRIPTION STARTS HERE
*
*
COPY "LB:[1,5]DEMLIB.LIB".
*
*
*
*
PROCEDURE DIVISION.
MAIN-CONTROL SECTION.
P1.
*               Attach the terminal.
        CALL "WTQIO" USING INUM, UN, UN.
*               Initialize and open the library.
        CALL "FINIT" USING IMPURE, ISIZE.
        CALL "FLCHAN" USING LUN.
        PERFORM STATUS-CHECK.
        CALL "FLOPEN" USING DEMLIB.
        PERFORM STATUS-CHECK.
*               Display first form.
P6.
        CALL "FCLRSH" USING FORM-FIRST.
        PERFORM STATUS-CHECK.


*               Show the menu form for operator to select the data
*               collection series.  Get the first form name from
*               named data.
P2.
        CALL "FGET" USING
         D-FIRST-CHOICE, STAT, N-FIRST-CHOICE.
        PERFORM STATUS-CHECK.
        MOVE  D-FIRST-CHOICE TO FIELD.
        MOVE SPACES TO ANSWER1.
        CALL "FNDATA" USING
         FIELD, ANSWER1.
        CALL "FSTAT" USING STAT.
        IF STAT NOT > 0
         CALL "FPUTL" USING ILL-CHOICE
         PERFORM STATUS-CHECK
         GO TO P2.
*
*               If form name is ".EXIT.", terminal operator is done.
*
```

```
        IF PART OF ANSWER1 = ".EXIT." GO TO LIB-CLOSE.
*
*               Get the output file name from named data and open it.
*
        MOVE D-FIRST-CHOICE TO DAT OF FIELDF.
        MOVE SPACES TO ANSWER2.
        CALL "FNDATA" USING
         DAT OF FIELDF, ANSWER2.
        PERFORM STATUS-CHECK.
        OPEN OUTPUT OUTPUT-FILE.
*
*               This is the data collection loop.
*
P4.
        MOVE ANSWER1 TO ANSWER3.
*               Show the form.
P3.
        CALL "FCLRSH" USING ANSWER3.
        PERFORM STATUS-CHECK.
*
*               Get data for current form and output it.
*
        MOVE SPACES TO POOL.
        CALL "FGETAL" USING POOL.
        PERFORM STATUS-CHECK.
        WRITE POOL.
*
*               Get name of next form.  If found, loop for more data.
*
        MOVE "NXTFRM" TO FIELD.
        CALL "FNDATA" USING
        FIELD, ANSWER3.
        PERFORM STATUS-CHECK.
        IF PART OF ANSWER3 NOT = ".NONE." GO TO P3.
*
*               End of the form series.  Show last to determine if
*               we're done or not.
*
        CALL "FCLRSH" USING FORM-LAST.
        PERFORM STATUS-CHECK.


P5.
        CALL "FGET" USING
         D-LAST-CHOICE, STAT, N-LAST-CHOICE.
        PERFORM STATUS-CHECK.
        MOVE D-LAST-CHOICE TO FIELD.
*
*               If response = "1", repeat data collection loop.
*
        IF FIELD = "1" GO TO P4.
*
*               Get named data corresponding to response.
*               Get field again if illegal response.
*               Close output file for valid response other than 1.
*
        CALL "FNDATA" USING FIELD, ANSWER3.
        CALL "FSTAT" USING STAT.
        IF STAT NOT > 0
         CALL "FPUTL" USING ILL-CHOICE
         PERFORM STATUS-CHECK
         GO TO P5.
        CLOSE OUTPUT-FILE.
*
*               If named data is ".EXIT.", terminal operator
*               is done, else display menu form again.
*
        IF PART OF ANSWER3 NOT = ".EXIT." GO TO P6.
```

```
*
*               Close form library and exit.
*
LIB-CLOSE.
        CALL "FLCLOS".
        PERFORM STATUS-CHECK.
        STOP RUN.


*
*               Output message and exit if I.O error returned from
*               Form Driver.  This is the only error expected in a
*               debugged application.
*
 STATUS-CHECK SECTION.
 SC1.
        CALL "FSTAT" USING STAT, STAT2.
        IF STAT > 0 GO TO SC2.
        MOVE STAT TO ERR-STAT.
        MOVE STAT2 TO ERR-STAT2.
        DISPLAY ERR1 AT LINE 1 AT COLUMN 1, ERASE TO END OF SCREEN.
        DISPLAY "Press RESUME to continue." AT LINE 3 AT COLUMN 1.
        CALL "WTRES".
        STOP RUN.
 SC2.
        EXIT.
```

7.3 TOOL KIT FORTRAN-77

```
C
C FORDEM.FTN
C
C
C                    COPYRIGHT (C) 1979 BY
C        DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
C
C
C MODULE:        FORDEM
C
C VERSION:       V01.00
C
C AUTHOR:
C
C DATE:          1-APRIL-79
C
C MODIFIED
C
C DATE:          2-MARCH-83      To run on the Professional
C
C
C FORTRAN demonstration program for FMS illustrating a
C simple form-driven, data entry application.
C
C Below is an example command file to build this demonstration program
C
C        SY:FORDEM/CP,SY:FORDEM/-SP=SY:FORDEM/MP
C        ;
C        TASK = FORDEM
C        ;
C        UNITS = 19
C        ;
C        ASG=TI:5:13:15
C        ASG=SY:1:2:7:8:9:10:11:12
C        ;
C        CLSTR=PROF77,POSRES,RMSRES:RO
C        ;
C        EXTSCT=MN$BUF:0          ;SINGLE CHOICE MENU
C        EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE
C        EXTSCT=HL$BUF:3410       ;HELP
C        EXTSCT=MS$BUF:3100       ;MESSAGE
C        EXTSCT=MM$BUF:0          ;MUTLI-CHOICE MENU
C        EXTSCT=FL$BUF:0          ;MULTI-CHOICE MENU
C        GBLDEF=MN$LUN:22         ;MENU
C        GBLDEF=HL$LUN:20         ;HELP
C        GBLDEF=MS$LUN:21         ;MESSAGE
C        GBLDEF=TT$LUN:15         ;TERMINAL I/O
C        GBLDEF=WC$LUN:23
C        GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
C        //
C


C Below is an example ODL file to build this demonstration program
C
C                .ROOT FORDEM-FDV-RMSROT-OTSROT-OTSALL
C        FDV:    .FCTR LB:[1,5]HLLFOR-LB:[1,5]FDV/LB
C        @LB:[1,5]PROF77
C        @LB:[1,5]RMSRLX
C                .END
C
C
C
```

```
        IMPLICIT INTEGER (A-Z)
        DIMENSION IMPURE (1000)
        BYTE RESP(3), FORM(7), FORM1(7), DNAM(3), FILE(30), DATA(255)
C
C Initialize impure area for Form Driver and open form library.
C
        CALL WTQIO (768,5,5)            !ATTACH THE TERMINAL
        CALL FINIT (IMPURE, 1000)
        CHAN=2
        CALL FLCHAN(CHAN)
        CALL ERROR (FLOPEN ('LB:[1,2]DEMLIB'))
C
C Display menu form.
C
 10     CALL ERROR (FCLRSH ('FIRST '))
C
C Get input from terminal. Get named data (name of first form in
C series or .EXIT.) corresponding to user's choice. If named data
C doesn't exist, input invalid.
C
 20     CALL ERROR (FGET (RESP, TERM, 'CHOICE'))
        IF (FNDATA (RESP, FORM1) .GT. 0) GOTO 30
        CALL FPUTL ('Illegal choice')
        GOTO 20
C
C Check for exit. If choice not exit, get name of corresponding
C file and open it for output.
C
 30     IF (SCOMP (FORM1, '.EXIT.') .NE. 0) GOTO 40
        CALL FLCLOS     ! CLOSE FORM LIBRARY
        STOP
 40     CALL CONCAT (RESP, 'F', DNAM)
        CALL FNDATA (DNAM, FILE)
        OPEN (NAME=FILE,UNIT=1,STATUS='NEW',INITIALSIZE=10)
C
C Display form and collect data; write data to output file.
C
 50     CALL SCOPY (FORM1, FORM, 6)
 60     CALL ERROR (FCLRSH (FORM))
        CALL ERROR (FGETAL (DATA))
        WRITE (1,70) (DATA(I), I=1,LENGTH(DATA))
 70     FORMAT (78A1) !DATA IS BROKEN INTO SEGMENTS FOR OUTPUT
C
C Get name of next form in series. Check for none.
C
        CALL FNDATA ('NXTFRM', FORM)
        IF (SCOMP (FORM, '.NONE.') .NE. 0) GOTO 60
C
C If last form in series done, display a menu form.
C Get input from terminal. Get named data corresponding
C to user's choice. If no named data, invalid input.
C
        CALL ERROR (FCLRSH ('LAST'))
 80     CALL ERROR (FGET (RESP, TERM, 'CHOICE'))
        IF (FNDATA (RESP, FORM) .GT. 0) GOTO 90
        CALL FPUTL ('Illegal choice')
        GOTO 80
C
C If choice = 1, repeat series.
C Else close output file; check for exit or go back to
C initial menu form.
C
 90     IF (RESP(1) .EQ. '1') GOTO 50
        CLOSE (UNIT=1)
        IF (SCOMP(FORM, '.EXIT.') .NE. 0) GOTO 10
        CALL FLCLOS     ! CLOSE FORM LIBRARY
        STOP
        END
```

```
        SUBROUTINE ERROR (RESULT)
C
C Output message and exit if I/O error returned from
C Form Driver. This is the only error expected in a
C debugged application.
C
        IMPLICIT INTEGER (A-Z)
C
        IF (RESULT .GT. 0) RETURN
        CALL FPUTL ('Fatal I/O Error')
        STOP
        END


        SUBROUTINE SCOPY (SRC, DST, LEN)
C
C Copy a string of a specified length
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


        INTEGER FUNCTION SCOMP (SRC1, SRC2)
C
C Compare two strings
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
```

```
        SUBROUTINE CONCAT (SRC1, SRC2, DST)
C
C Concatenate two strings into a third
C
C SRC1 = first source string ended by a zero
C SRC2 = second source string ended by a zero
C DST = destination string ended by a zero
C
        BYTE SRC1(1), SRC2(1), DST(1)
C
C Copy the first string into destination
C
        J = 1
        I = 1
10      IF (SRC1(I) .EQ. 0) GOTO 20
        DST(J) = SRC1(I)
        J = J + 1
        I = I + 1
        GOTO 10
C
C Now for second string to destination
C
20      I = 1
30      DST(J) = SRC2(I)
        IF (SRC2(I) .EQ. 0) GOTO 40
        J = J + 1
        I = I + 1
        GOTO 30
C
C Return
C
40      RETURN
        END


        SUBROUTINE INSERT (SRC, DST, POS)
C
C Replace a portion of one string with another
C
C SRC = source string ended by a zero
C DST = destination string ended by a zero
C POS = position in destination for source string contents
C
        BYTE SRC(1), DST(1)
        INTEGER POS
C
C Scan the destination string for its end
C
        J = 1
10      IF (DST(J) .EQ. 0) GOTO 20
        J = J + 1
        GOTO 10
C
C Copy source into destination at position given
C
20      I = 1
30      IF (SRC(I) .EQ. 0) GOTO 40
        DST(I+POS-1) = SRC(I)
        I = I + 1
        GOTO 30
C
C End destination string if source extends it and return
C
40      IF (I .GT. J) DST(J) = 0
        RETURN
        END
```

```
        INTEGER FUNCTION INDEX (SRC, STR)
C
C Find position of one string in another
C
C SRC = source string
C STR = target string
C
C Value of function is zero if not found,
C or position of first character of STR in SRC if found
C
        BYTE SRC(1), STR(1)
C
C Look for STR in SRC until end of SRC
C
        J = 0
10      J = J + 1
        I = 0
        IF (SRC(J) .EQ. 0) GOTO 30
C
C If end of STR then success
C If not match look at next position in SRC
C
20      IF (STR(I+1) .EQ. 0) GOTO 40
        IF (SRC(J+I) .NE. STR(I+1)) GOTO 10
        I = I + 1
        GOTO 20
C
C Return failure
C
30      INDEX = 0
        RETURN
C
C Return success, position of string
C
40      INDEX = J
        RETURN
C
        END


        INTEGER FUNCTION LENGTH (STR)
C
C Return length of string ended by a zero
C
C STR = string ended by a zero
C
C Value of the function is the length of the string without the zero
C
        BYTE STR(1)
C
C Scan for the zero byte
C
        I = 1
10      IF (STR(I) .EQ. 0) GOTO 20
        I = I + 1
        GOTO 10
C
C Return the length of the string
C
20      LENGTH = I - 1
        RETURN
        END
```

7.4 TOOL KIT MACRO-11

```
        .TITLE  MACDEM  - FMS DEMONSTRATION SUBROUTINE

;
; MACDEM.MAC
;
;
;                       COPYRIGHT (C) 1979 BY
;             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
;
;
; MODULE:       MACDEM
;
; VERSION:      V01.00
;
; AUTHOR:
;
; DATE:         19-NOVEMBER-79
;
; MODIFIED:     SGD001
;
; DATE:         14-APRIL-1983  -- TO RUN ON THE PROFESSIONAL
;
;       Assembly and Compile instructions:      (VAX)
;
; To assemble type:
; MCR PMA <RET>
; PMA> MACDEM,MACDEM/-SP=LB:[1,5]FMSMAC/ML,LB:[1,5]RMSMAC/ML,DEV:[UIC]MACDEM
;
;       To compile type:
;       MCR PROTKB
;       PAB> @MACDEM.CMD
;
;       Assembly and Compile instructions:      (RSX)
;
; To assemble type:
; RUN $PMA <RET>
; PMA> MACDEM,MACDEM/-SP=LB:[1,5]FMSMAC/ML,LB:[1,5]RMSMAC/ML,DEV:[UIC]MACDEM
;
;       To compile type:
;       RUN $PROTKB
;       PAB> @MACDEM.CMD
;
        .ENABL  LC                          ; Allow lower case source text

        .MCALL  $FDV,$FDVDF                 ; Identify Form Driver macro calls
        .MCALL  QIOW$S,EXIT$S,DIR$,ALUN$    ; RSX I/O related macros
        .MCALL  FAB$B, RAB$B, POOL$B        ; RMS related macros
        .MCALL  $STORE, $COMPARE, $CREATE   ;
        .MCALL  $CONNECT, $DISCONNECT       ;
        .MCALL  $PUT, $CLOSE, ORG$          ;

        $FDVDF                              ; Init the Form Driver definitions

;
; Equated symbols
;

        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
CHCNAM: .ASCII  /CHOICE/                ; ASCII field name
NXTNAM: .ASCII  /NXTFRM/                ; ASCII named data field name
LIBNAM: .ASCIZ  /LB:[1,2]DEMLIB/        ; ASCII library 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

FRMNAM: .BLKW   3                       ; Area for form names
SAVNAM: .BLKW   3                       ; Save area for a form name
IMPURE: .WORD   ISIZ                    ; Form Driver impure area
        .BLKB   ISIZ-2

;
; I/O section
;
        .EVEN
FABADD:
        FAB$B                           ; Allocate RMS FAB
        F$DEQ   2                       ; Default file extension size
        F$ALQ   2                       ; Allocation size for the file
        F$FOP   FB$SUP                  ; Create new file
        F$FAC   FB$PUT                  ; File access operations
        FAB$E                           ; End FAB declarations

        .EVEN
RABADD:
        RAB$B                           ; Allocate RMS RAB
        R$FAB   FABADD                  ; Connect to FAB address
        R$RAC   RB$SEQ                  ; Record access is sequential
        RAB$E

        .EVEN
        POOL$B                          ; Begin Pool declarations
        P$BDB   2                       ; Allow two buffer desc blocks
        P$FAB   1                       ; Only one file will be open
        P$RAB   1                       ; Need only one RAB
        P$BUF   512.                    ; I/O Buffer space
        POOL$E

        .EVEN
RMS$LUN:
        ALUN$   OU$CHN, SY, 0           ; Assign a LUN to the Disk
        .EVEN
        ORG$    SEQ,<CRE,PUT>           ; Define RMS needed functions

        .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
DEMO:
        DIR$    #RMS$LUN                ; Assign the LUN 0 to the Disk
        QIOW$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$CHN(R1)       ; Set I/O channel for FDV
        MOV     #IMPURE,F$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 with error

FIRST:  $FDV    FNC=CSH,NAM=#FSTNAM     ; Show menu form
        CALL    ERREX                   ; Exit if error

10$:    $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
        $FDV    FNC=DAT,NAM=#VAR1       ; Get named data with the name being
                                        ;   the response to 'CHOICE'
        CMP     STAT,#FS$SUC            ; Was get successful?
        BEQ     20$                     ; Continue if ok
        $FDV    FNC=LST,VAL=#MSG1,LEN=#-1 ; Else print message on line 24
        BR      10$                     ; 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
         MOVB    #'F,VAR1+1              ; Make 2nd letter = F
         MOV     #VAR1,R1                ; R1 = addr of 6-byte block
         $FDV    FNC=DAT,NAM=R1          ; Get named data at VAR1

         MOV     #FABADD,R3              ; Move the FAB address to R4
         MOV     #RABADD,R4              ; Move the RAB address to R4
         $STORE  F$VAL(R0),FNA,R3        ; Move the addr of the file name
         $STORE  F$LEN(R0),FNS,R3        ; Move the file name size
         $STORE  #OU$CHN,LCH,R3          ; Set the LUN in the FAB
         $CREATE R3                      ; Create the file
         $COMPARE #SU$SUC,STS,R3         ; Check RMS Status
         BEQ     40$                     ; Continue if status = 1
         CALL    LEAVE                   ; Leave on I/O error
```

```
40$:    $CONNECT R4                     ; Connect the RAB to the FAB
        $COMPARE #SU$SUC,STS,R4         ; Check RMS Status
        BEQ     60$                     ; Continue if ok
        CALL    LEAVE                   ; Leave on I/O error

60$:    $FDV    ARG=#ARGLST,FNC=CSH,NAM=#FRMNAM
        CALL    ERREX                   ; Exit with error

        $FDV    FNC=ALL                 ; Get all data from form
        CALL    ERREX                   ; Exit with 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=CSH,NAM=#LSTNAM
        CALL    ERREX                   ; Exit with error

80$:    $FDV    FNC=GET,NAM=#CHCNAM
        CALL    ERREX                   ; Exit with error

        MOV     F$VAL(R0),R1            ; R1 = adr of answer
        CMPB    (R1),#'1                ; Is it = 1
        BNE     90$
        MOV     #SAVNAM,R1              ; R1 = source name
        MOV     #FRMNAM,R2              ; R2 = dest name
        .REPT   3
            MOV  (R1)+,(R2)+            ; Move name
        .ENDR
        BR      60$                     ; Get more data

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::
        MOV     #FABADD,R3              ; Move the addr of the FAB to R3
        $DISCONNECT R4                  ; Disconnect the access stream
        $COMPARE #SU$SUC,STS,R4         ; Check RMS Status
        BEQ     95$                     ; Branch if Status OK
        CALL    LEAVE                   ; Else I/O error
95$:    $CLOSE  R3                      ; Close output file
        MOV     #EXTNAM,R1              ; Name of exit named data
        MOV     #ARGLST,R0              ; Get ARGLST
        MOV     F$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=CLS                 ; Close form library
        BR      EXIT                    ; And exit
```

```
;
; Routine to check for error return from Form Driver.
; Print message and exit on error.
;

ERREX:  CMP     STAT,#FS$SUC            ; Was call ok?
        BNE     LEAVE
        RETURN
LEAVE:  $FDV    ARG=#ARGLST
        $FDV    FNC=LST,VAL=#MSG2,LEN=#-1 ; Print message on line 24
EXIT:   EXIT$S

;
; Subroutine to store data in output file
;

SAVDAT: $STORE  F$VAL(R0),RBF,R4        ; Move the addr data to the RAB
        $STORE  F$LEN(R0),RSZ,R4        ; Move the len of data to RAB
        $COMPARE #0,RSZ,R4              ; See if the data length is zero
        BEQ     10$                     ; If not return
        $PUT    R4                      ; Store away the string of data
        $COMPARE #SU$SUC,STS,R4         ; Check the RMS Status
        BEQ     10$                     ; Branch if equal
        CALL    LEAVE                   ; Leave on I/O error
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

        .END    DEMO
```

```
;       MACDEM.CMD
;
;       Command file for FMS - MACRO Demonstration program
;
MACDEM/CP/-FP,MACDEM/-SP=MACDEM/MP
TASK = MACDEM
UNITS = 8
ASG = TI:5
ASG = SY:1:2:3:4:6:7:8
CLSTR = FMSRES,POSRES,RMSRES:RO

EXTSCT=MN$BUF:0          ;STATIC SINGLE CHOICE MENU
EXTSCT=DM$BUF:0          ;DYNAMIC SINGLE CHOICE MENU
EXTSCT=HL$BUF:3500       ;HELP TEXT/MENU
EXTSCT=MM$BUF:0          ;MUTLI-CHOICE MENU
EXTSCT=FL$BUF:0          ;FILE SELECTION/SPECIFICATION
GBLDEF=MN$LUN:2          ;MENU FRAME FILE
GBLDEF=HL$LUN:3          ;HELP FRAME FILE
GBLDEF=MS$LUN:4          ;MESSAGE FRAME FILE
GBLDEF=TT$LUN:5          ;TERMINAL I/O
GBLDEF=MB$LUN:6          ;MESSAGE/STATUS DISPLAY
GBLDEF=WC$LUN:7          ;DIRECTORY SEARCHES FOR OLDFIL AND NEWFILE
GBLDEF=TT$EFN:1          ;I/O EVENT FLAG
//

;       MACDEM.ODL
;
;       ODL file for the FMS - MACRO Demonstration program.
;
        .ROOT USER-RMSROT
USER:   .FCTR MACDEM - LB:[1,5]SYSLIB/DL
@LB:[1,5]RMSRLX
        .END
```

7.5 TOOL KIT PASCAL

When using PASCAL with PRO/FMS-11, follow these instructions:

- **Source Code.** PRO/FMS-11 calls for PASCAL are in the Tool Kit file FMS.PAS in directory LB:[1,5]. Use the %INCLUDE directive to include the file in your PASCAL source code:

```
{ Include PRO/FMS Procedures }
%Include 'LB:[1,5]FMS.PAS'
```

- **Calling Sequence.** FMS.PAS consists of SEQ11 procedures, which pass all parameters by reference. To determine the appropriate calling sequences for your application, refer to FMS.PAS and the section in the *Tool Kit PASCAL User's Guide* on interfaces between Tool Kit Pascal and P/OS and other Tool Kit Software.

NOTE

Pad PASCAL form and field names with six spaces. Otherwise, the Form Driver cannot access the form or field.

- **Parameters.** There are no optional parameters for PASCAL calls to PRO/FMS-11. You must include all parameters. For example, in FMS.PAS, declare FCLRSH as a SEQ11 procedure with two parameters. If you want to use the FCLRSH call, but you do not want to specify a new value for the starting line, pass 0 as the first parameter. The form displays at the starting line specified.

- **Indexes.** When using an FMS.PAS routine that allows indexes, you must specify an index for the variable. When you access the field, assign it an index of 1. For example:

```
index := 1;
field := 'CHOICE';
FGET (response, terminator, field, index);
```

- **Variables.** PASCAL can pass only variables when calling routines declared in FMS.PAS. For example, the following source statement would fail:

```
FCLRSH("FIRST ",0);
```

The correct calling sequence for FCLRSH is:

```
VAR Form_Name     : Packed Array [1..6] Of Char;
    Starting_Line : Integer;

{       Main Program       }

Form_Name     := 'FIRST ';
Starting_Line := 1;
FCLRSH(Form_Name,Starting_Line);
```

If you want to pass constants as parameters, edit FMS.PAS and assign the appropriate formal parameters the READONLY attribute. For example, if you edited FMS.PAS so that both FCLRSH parameters were READONLY, you could use the call:

```
FCLRSH('FIRST',1);
```

See the *PASCAL User's Guide* for details on interfaces between Tool Kit Pascal and PO/S and other Tool Kit software.

- **Library File Specification.** You must terminate a library file specification with a NUL character. Without it, PRO/FMS-11 cannot access the library file. For example, this is a correct PASCAL library file specification:

```
VAR File_Spec : Packed Array [1..15] Of Char;

{       Main Program       }

File_Spec := 'LB:[1,2]DEMLIB'(0);

FLOPEN(File_Spec);
```

- **FPUTL.** You must terminate data sent to the FPUTL call with a NUL character. Without it, random data might appear on the screen. For example, this PASCAL sequence would successfully pass data to FPUTL:

```
VAR Message : Packed Array [1..16] Of Char;

{       Main Program       }

Begin
Message := 'Example message'(0);
FPUTL(Message);
 End.
```

- **Impure Area and Data String Restrictions.** Using PASCAL with FMS.PAS, you can define an impure area as large as 1500 bytes and pass a data string of up to 1500 characters. If you need larger values, increase the variable MAX_FMS_PAR_LEN in FMS.PAS.

Add one byte to the size of your buffer where any call to the FMS Form Driver produces data returned to your program as a buffer. The Fortran Interface places a null byte at the end of the string returned to the caller to signify the end of the string. If you do not add the extra byte, the PRO/FMS-11 Form Driver overwrites part of your data space or program.

For example,if the field is one character long, the response buffer must be two characters long.

The following is a sample PRO/FMS-11 program in PASCAL, using FMS.PAS.

```
PROGRAM PASDEM (INPUT,OUTPUT);

{       PASDEM.PAS

                Copyright (C) 1983 By
        Digital Equipment Corporation, Maynard, Mass.


        Module: PASDEM

        Version V01.00

        Author:
        Date:   27-Apr-1983


        PASCAL Demonstration program for PRO/FMS illustrating a
        simple Form-Driven, data entry application.

        Below is an example command and odl file to build
        this demonstration program.


        ;PASDEM.CMD

        SY:PASDEM/CP/-FP,PASDEM/MA/-SP=SY:PASDEM/MP
        CLSTR=PASRES,POSRES,RMSRES,DAPRES:RO
        TASK=PASDEM
        UNITS  = 20
        ASG    = TI:5:13:15
        ASG    = SY:6:7:8:9:10:11:12
        EXTSCT = MN$BUF:0
        EXTSCT = HL$BUF:3410
        GBLDEF = MS$LUN:21
        GBLDEF = WC$LUN:23
        GBLDEF = HL$LUN:20
        GBLDEF = MN$LUN:22
        GBLDEF = TT$LUN:15
        GBLDEF = TT$EFN:1
        //
```

```
        Example ODL file:

        ;PASDEM.ODL

                .ROOT   USER - PASCAL

        USER:   .FCTR   SY:PASDEM

        PASCAL: .FCTR   LB:[1,5]PASLIB/LB-FDV-RMSROT
        FDV:    .FCTR   LB:[1,5]HLLFOR-LB:[1,5]FDV.OLB/LB
        @LB:[1,5]DAPRLX
                .END

        Include PRO/FMS Procedures       }

        %Include 'LB:[1,5]FMS.PAS'

{       Declare types and variables      }

TYPE
        Impure      = PACKED ARRAY [1..1000] Of Integer;
        Forms       = PACKED ARRAY [1..6]    Of Char;
        File_Spec   = PACKED ARRAY [1..15]   Of Char;
        Buffer      = PACKED ARRAY [1..225]  Of Char;
        Out_Line    = PACKED ARRAY [1..41]   Of Char;
        Named_Data  = PACKED ARRAY [1..66]   Of Char;

VAR
        QIO_Function,                   { Qio function code              }
        TT_LUN,                         { Terminal LUN                   }
        TT_EFN,                         { I/O event flag                 }
        Index,                          { FMS field index                }
        Length,                         { Length of data                 }
        Channel,                        { FMS Library channel            }
        Terminator,                     { Form terminator                }
        Starting_Line,                  { Starting line for forms        }
        Status_1, Status_2,             { Status values of FMS calls     }
        Impure_Size:   Integer;         { FMS Impure area                }
        Impure_Area:   Impure;          { Size of FMS Impure area        }
        Library:       File_Spec;       { Forms library                  }
        Field,                          { FMS field name                 }
        Response,                       { User's response                }
        Next_Form,                      { Next form to display           }
        Current_Form:  Forms;           { Current form to display        }
        Message:       Out_Line;        { Message for FPUTL call         }
        All_Data:      Buffer;          { Storage for all data in a form }
        Name_Data:     Named_Data;      { Storage for named data         }
        More_Data:     Boolean;         { Flag for more data             }
        Out_File:      Text;            { File variable for outfile      }

{     ***** Procedure to wait for the RESUME key *****

        This procedure is called in the event that the forms library can not
    be opened. This procedure calls the routine WTRES in the P/OS callable
    library.                                                              }

PROCEDURE WTRES; SEQ11;


{     ***** Procedure to attach the terminal *****

        Call this procedure to attach the terminal. The Form Driver
     needs the terminal attached. Call the FORTRAN routine WTQIO in
     SYSLIB.                                                              }

PROCEDURE WTQIO(VAR QIO_Function: Integer;  {Function code for QIO }
                VAR TT_LUN: Integer;        {I/O Channel           }
```

```
                VAR TT_EFN: Integer          {I/O Channel           }
               ); SEQ11;

{      ***** Procedure to move data in one variable to another *****

         Call this procedure to move character data from one variable
      to another variable. This is useful if the two variables differ in
length. Three parameters pass. Copy the two variables and the number
of characters from the first variable to the second variable.     }

PROCEDURE MOVE(VAR Var1: Packed Array[LB1..UP1 : Integer] Of Char;
               VAR Var2: Packed Array[LB2..UP2 : Integer] Of Char;
               VAR Length : Integer
              );

VAR
        I : Integer;

Begin

        I := 1;
        While I <= Length Do
           Begin
             Var2[I] := Var1[I];
             I := I + 1
           End

End;

{            End of Procedure MOVE              }

{     **********  Main Program **********      }

Begin

{      ***** Initialize variables *****      }

        Index := 0;
        Channel := 7;
        Impure_Size := 1000;
        Library := 'LB:[1,2]DEMLIB'(0);
        QIO_Function := 768;
        TT_LUN := 5;
        TT_EFN := 5;
        Starting_Line := 1;
        Current_Form := '       ';

{      ***** Initialize FMS Impure Area and Open Library *****      }

        WTQIO(QIO_Function,TT_LUN,TT_EFN);       {Attach the terminal        }

        FINIT(Impure_Area,Impure_Size,Status_1); {Iniatilize FMS Impure Area  }
        FLCHAN(Channel);                         {Set the library channel    }
        FLOPEN(Library);                         {Open demonstration library }
        Writeln(CHR(27),'[2J');                  {Clear the screen           }

{       Display menu form for operator to select the data collection
        series. This will continue until the operator chooses the exit
        selection from either the form called FIRST or the form called LAST  }

        While Current_Form <> '.EXIT.' Do

        Begin

           Current_Form := 'FIRST ';
           FCLRSH(Current_Form,Starting_Line);   {Display the first form    }
           FSTAT(Status_1,Status_2);             {Check the status          }
```

```
If Status_1 = 1 Then
  Begin
    Field := 'CHOICE';
    Status_1 := 0
  End
Else                                  {Else display error message }
  Begin                               {and exit.                  }
    Status_1 := 1;
    Next_Form := '.EXIT.';
    Length := 6;
    MOVE(Next_Form,Name_Data,Length);
    Writeln('Error opening library file, Press RESUME to continue.');
    Writeln;
    WTRES                             {Wait for the operator to   }
  End; {IF}                           {read error message         }

While Status_1 <> 1 Do

Begin
   FGET(Response,Terminator,Field,Index);
   FNDATA(Response,Name_Data);
   FSTAT(Status_1,Status_2);
   If Status_1 < 0 Then
   Begin
     Message := 'Illegal Choice                          '(0);
     FPUTL(Message)
   End; {IF}

End; {While Status}

Length := 6;
MOVE(Name_Data,Current_Form,Length);

If Current_Form <> '.EXIT.' Then
Begin
   Field := Response;
   Field[2] := 'F';
   FNDATA(Field,Name_Data);
   Open(Out_File,
        File_name:=Name_Data,
        History:=New,
        Record_length:=255
        );

   More_Data := TRUE;
   While More_Data Do
   Begin;
      Next_Form := Current_Form;
      While Next_Form <> '.NONE.' Do
      Begin
        FCLRSH(Next_Form,Starting_Line);
        FGETAL(ALL_Data,Terminator);
        Write(Out_File,All_Data);
        Field := 'NXTFRM';
        FNDATA(Field,Name_Data);
        MOVE(Name_Data,Next_Form,Length);
      End; {While Next Form}

      Status_1 := 0;
      Field := 'CHOICE';
      Next_Form := 'LAST  ';
      FCLRSH(Next_Form,Starting_Line);
      While Status_1 <> 1 Do
      Begin
        FGET(Response,Terminator,Field,Index);
        FSTAT(Status_1,Status_2);
        FNDATA(Response,Name_Data);
```

```
                    FSTAT(Status_1,Status_2);
                    If Status_1 < 0 Then
                    Begin
                       Message := 'Illegal Choice                    '(0);
                       FPUTL(Message)
                    End {IF}
                  End; {End While Status}

                  If Response[1] = '2' Then
                  Begin
                    More_Data := FALSE ;
                    Close(Out_File)
                  End; {IF}

                  If Response[1] = '3' Then
                  Begin
                    More_Data := FALSE;
                    Close(Out_File);
                    Current_Form := '.EXIT.'
                  End; {IF}

               End; {While More_Data }

            End; {IF}

         End; { While Current_Form }

         FLCLOS; { Close FMS library file }

End.
```

INDEX