# PRO/FMS-11 Documentation Supplement

Order No. AA-P103B-TK

**May 1983**

This supplement describes the differences between PRO/FMS-11 and FMS/RSX-11. PRO/FMS-11 may be used with the Professional 300 Series Developer's Tool Kit to write application software to run under the Professional Operating System (P/OS).

| | |
|---|---|
| OPERATING SYSTEM AND VERSION: | VAX/VMS V3.2 or later<br>RSX-11M V4.1 or later<br>RSX-11M-PLUS V2.1 or later |
| SOFTWARE VERSION: | Professional Developer's Tool Kit V1.5<br>PRO/FMS-11 V1.0 |

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754

First Printing December 1982
Revised May 1983

# CONTENTS

# PRO/FMS-11 DOCUMENTATION SUPPLEMENT

## 1.0 RELATED DOCUMENTATION

This manual supplements two manuals: *FMS-11/RSX Software Reference Manual* and *FMS-11/RSX Release Notes*. Both these manuals are included in the *Tool Kit Documentation Directory*. The *FMS-11/RSX Mini-Reference* is not included in the Tool Kit documentation set, but may also be useful when you are developing PRO/FMS-11 applications.

## 2.0 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 runs on the Professional Developer's Tool Kit host systems: RSX-11M/11M-PLUS and VAX/VMS. This supplement describes the differences between FMS-11 applications designed for minicomputer/VT100 systems and PRO/FMS-11 applications designed for the Professional.

The material here is intended for experienced FMS-11 programmers. If you have not previously programmed with FMS-11, you should start with the FMS/RSX-11 documentation.

## 3.0 PRO/FMS-11 PROGRAM DEVELOPMENT

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

- ☐ Tool Kit BASIC-PLUS-2
- ☐ Tool Kit COBOL-81
- ☐ Tool Kit FORTRAN-77
- ☐ Tool Kit MACRO-11
- ☐ Tool Kit PASCAL

For specific restrictions and examples, turn to the section, "Sample PRO/FMS-11 programs" at the end of this manual.

Two PRO/FMS-11 files, DEMLIB.FLB and FMSDBG.MSG, are supplied on diskette with the Tool Kit. You will need these files to run the sample programs and debug your own applications. To copy them to your Professional, follow the steps in this manual's section "Installation Procedures."

The program development cycle for PRO/FMS-11 applications is as follows:

1. Using a VT100, or a Professional in Terminal Emulation Mode, create forms on the host system with the PRO/FMS Forms Editor (PROFED). To invoke PROFED on RSX, type:

   ```
   >RUN $PROFED
   ```

   On VMS, type:

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

2. Create a form library on the host system with the PRO/FMS Form Utility (PROFUT). To invoke PROFUT on RSX, type:

   ```
   >RUN $PROFUT
   ```

   On VMS, type:

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

3. Write a source program; include the necessary Form Driver calls in the source code.

   Note: Use the debug version of the Form Driver (FDVDBG.OLB) to debug at the Professional. Use the non-debug version (FDV.OLB) when program is error-free.

4. Compile (or assemble) the program.
5. Following the instructions in this manual, include the Form Driver in the overlay descriptor (.ODL) file.
6. Task build the program.
7. Write the application Installation file.
8. Copy the form library, task image, and application installation file to the Professional.
9. Install the application onto a P/OS menu.
10. Run the program.

Figure 1 illustrates the development cycle.

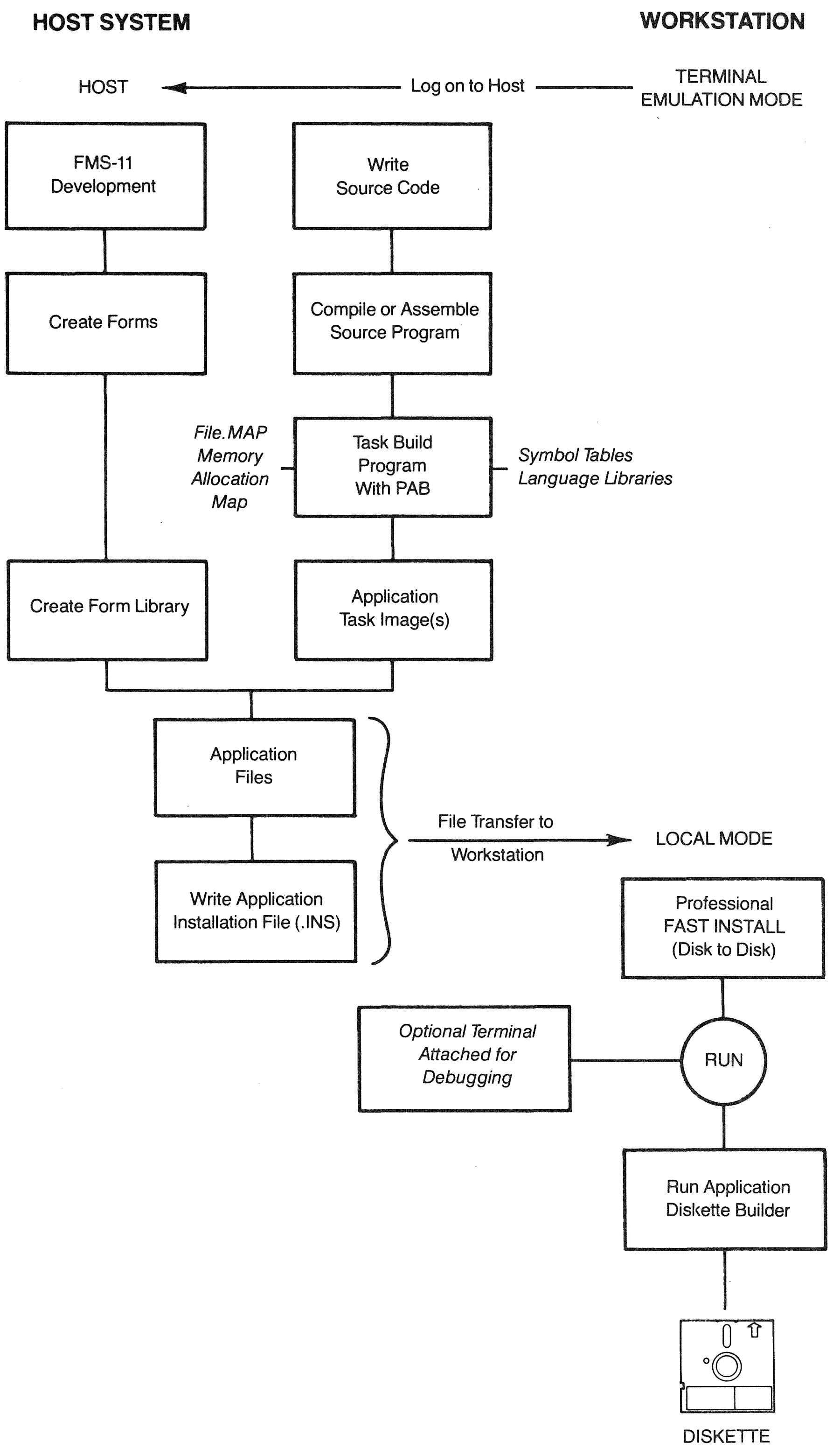
HOST SYSTEM
WORKSTATION
HOST
Log on to Host
TERMINAL EMULATION MODE
FMS-11 Development
Write Source Code
Create Forms
Compile or Assemble Source Program
File.MAP Memory Allocation Map
Task Build Program With PAB
Symbol Tables Language Libraries
Create Form Library
Application Task Image(s)
Application Files
Write Application Installation File (.INS)
File Transfer to Workstation
LOCAL MODE
Professional FAST INSTALL (Disk to Disk)
Optional Terminal Attached for Debugging
RUN
Run Application Diskette Builder
DISKETTE

Figure 1 PRO/FMS-11 Development Cycle

## 4.0 THE FORMS EDITOR

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

### 4.1 Terminal Emulation

To run PROFED in terminal emulation, you must set your terminal to VT102 emulation mode. If you want to use DEC multinational 8-bit mode characters, then you must also:

- ☐ Set the Professional to Terminal Emulator 8-bit mode.
- ☐ Set the host to 8-bit mode. On RSX systems use the command:

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

  On VMS systems use the command:

  ```
  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 can include characters 241-277 octal.
- ☐ Background text can include the entire DEC Multinational Character Set.
- ☐ During application execution, PROFED forms (named data, form wide attributes, and field attributes) will accept the DEC Multinational Character Set as input where any displayable character is requested.

Note that the DEC Multinational Character Set uses 8-bit codes: if you want to use 8-bit codes in form descriptions, you must create the forms on a Professional. (See the *Terminal Subsystems Manual* for more information on the DEC Multinational Character Set.) Otherwise you can create forms on a VT100, using the full printable ASCII character set.

### 4.2 Attribute Differences

Three attributes operate differently with PRO/FMS-11 on the Professional from FMS/RSX-11 on a VT100:

**Bold field**—supported in wide-screen (132-column) mode.

**Bold reverse video field**—Not recommended.

**Blink field**—Can detract from PRO/FMS-11 performance if used excessively.

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

## 5.0 THE FORM DRIVER

The form library file specifications must have the following format:

```
device:[directory]file.typ[;version]
```

The directory must be enclosed in square brackets. Only the version is optional. To determine the device and directory where the application resides, use PROLOG services (a P/OS service) to translate the logical APPL$DIR. See the *P/OS System Reference Manual* for more details on PROLOG services.

### 5.1 The Professional Keyboard And The Form Driver

PRO/FMS-11 uses different keys for some field terminators and interactive functions involving the Form Driver. The following tables list these differences.

Table 1
Keyboard Differences—All Regions

| *Function* | *VT100* | |
|---|---|---|
| Enter form | ENTER/RETURN | DO, ENTER/RETURN |
| Move to next field | TAB | F12 |
| Move to previous field | BACKSPACE | F11 |
| Cursor left | ← | ← |
| Cursor right | → | → |
| Erase character | DELETE | ⌫ |
| Erase field | LINEFEED | REMOVE |
| Insert/Overstrike | PF1 | F13 |
| Help | PF2 | HELP |
| Repaint screen | CTRL/W | F20 |

Table 2
Keyboard Differences—Scrolled Regions

| *Function* | *VT100* | *Professional* |
|---|---|---|
| Move to previous line | ↑ | ↑ |
| Move to next line | ↓ | ↓ |
| Exit field backward | PF1 | F17 |
| Exit field forward | PF2 | F18 |

When a help form is displayed, the **RESUME** key can be used instead of the **DO** key, and the **NEXT SCREEN** key can be used instead of **HELP**.

## 5.2 Task Building with the Form Driver Object Library

The Form Driver object library must be task built with your application program.

> Note: Because the Form Driver requires P/OS User Interface Service Routines support, you must perform the procedure described in the section in the *Tool Kit User's Guide* on the Professional Application Builder.

For example, if your source code is in BASIC-PLUS-2, the BASIC-PLUS-2 compiler will generate a command file (filename.CMD) and an overlay description file (filename.ODL) when you enter the BUILD command. You must edit the command file and the overlay description file to include the Form Driver object library.

**5.2.1 Editing the Command File**—You must edit the command (.CMD) file for your PRO/FMS-11 application for the language you are using. See your language documentation for details. The command (.CMD) file generated by the BASIC-PLUS-2 compiler for an application named "TEST1" would look something like this once it had been edited for BASIC-PLUS-2:

```
SY:TEST1/CP=SY:TEST1/MP
TASK = task-name
UNITS = 18
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
EXTTSK= 952
CLSTR = PBESML,RMSRES,POSRES:RO
EXTSCT  = MN$BUF:4540    ; static single choice menu
EXTSCT  = DM$BUF:4540    ; dynamic single choice menu
EXTSCT  = MM$BUF:1000    ; multi-screen menu
EXTSCT  = HL$BUF:3410    ; help text/menu
EXTSCT  = FL$BUF:4310    ; file selection/specification
GBLDEF  = MN$LUN:20      ; menu frame file
GBLDEF  = HL$LUN:21      ; help frame file
GBLDEF  = MS$LUN:16      ; message frame file
GBLDEF  = TT$LUN:15      ; terminal I/O
GBLDEF  = TT$EFN:1       ; terminal I/O event flag
GBLDEF  = WC$LUN:22      ; directory searches for OLDFIL and NEWFIL
                         ; routine or callable print services
//
```

You must also edit the command file to use it with PRO/FMS-11. Follow these steps:

- ☐ FDV uses Logical Unit 5 for input and output to the terminal. Find the lines beginning with "ASG" and edit them to read:

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

- ☐ Set the extend section (EXTSCT) command as follows:

  ```
  EXTSCT = HL$BUF:3410     ; help text/menu
  ```

  If your application uses the P/OS Help services (described in the next section of this supplement), compute your frame size and set the extension to this value if it is larger than the current value. See Chapter 2 of the *Tool Kit User's Guide* for information on calculating frame size.

The fully edited command file for "TEST1" would look like this (changed lines are shaded):

```
SY:TEST1/CP=SY:TEST1/MP
TASK = task-name
UNITS = 18
ASG = TI:13:15:5
ASG = SY:6:7:8:9:10:11:12
EXTTSK= 952
CLSTR = PBESML,RMSRES,POSRES:RO
EXTSCT   = MN$BUF:4540    ; static single choice menu
EXTSCT   = DM$BUF:4540    ; dynamic single choice menu
EXTSCT   = MM$BUF:1000    ; multi-screen menu
EXTSCT   = HL$BUF:3500    ; help text/menu
EXTSCT   = FL$BUF:4310    ; file selection/specification
GBLDEF   = MN$LUN:20      ; menu frame file
GBLDEF   = HL$LUN:21      ; help frame file
GBLDEF   = MS$LUN:16      ; message frame file
GBLDEF   = TT$LUN:15      ; terminal I/O
GBLDEF   = TT$EFN:1       ; terminal I/O event flag
GBLDEF   = WC$LUN:22      ; directory searches for OLDFIL and NEWFILE
                          ; routine or callable print services
//
```

**5.2.2 Editing The Descriptor File**—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 overlay descriptor (.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, you would change the first line from this:

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

To this:

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

In addition, you must add a new .FCTR line after the LIBR: line of your .ODL file:

☐ 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: Your edit to a COBOL-81 file will depend on whether your program passes variables by descriptor or by reference. Look at the section "Tool Kit COBOL-81" in this manual for more information and sample programs. If variables are passed by descriptor and you are using the non-debug version of the Form Driver, add the line:

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

If variables are passed by descriptor and you are using the debug version the Form Driver, add the line:

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

If variables are passed by reference and you are using the non-debug version of the Form Driver, add the line:

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

If variables are passed by reference and you are using the debug version of the Form Driver, add the line:

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

The original .ODL file for a BASIC-PLUS-2 application named "TEST1" would look like this:

```
         .ROOT BASIC2-RMSROT-USER,RMSALL
USER:    .FCTR SY:TEST1-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 are shaded):

```
         .ROOT BASIC2-RMSROT-USER-FDV,RMSALL
USER:    .FCTR SY:TEST1-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:TEST1-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 at the end of this manual for sample edited .ODL files.

## 6.0 HELP

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

When the end user first presses the **HELP** key, help is displayed from the sources in the following order:

1. A one-line help message for current field
2. The Help form specified by current form
3. The P/OS help frame specified in the Named Data section of the current form

When the end user presses the **HELP** key again or presses the **NEXT SCREEN** key, help is displayed from the sources in this order:

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

To provide P/OS Help services with PRO/FMS-11, follow these steps:

1. Use the Frame Development Tool (FDT) to create help frame files.
2. Run the PRO/FMS-11 Forms Editor (PROFED) and retrieve the form. Enter the NAME command. PROFED will display the Named Data Entry Form.
3. In Named Data Entry Form, enter ".HELP." in the Name field and the frame identifier (frameid) in the Data field.
4. Edit your source code so that it includes calls to P/OS Help services to open the Help file before the Form Driver is called, and to close the Help file before the program exits.
5. Make sure you have edited the command file according to Section 5.2.1 of this supplement.
6. 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
```

You may use a maximum of 60 forms per library, with up to 4 libraries open at one time.

See the *Tool Kit User's Guide* for a description of the Frame Development Tool and information on creating Help definition files.

## 7.0 FLOPEN

FLOPEN, the FMS command to open a specified form library file, always returns a successful completion message, regardless of the call's actual success or failure. Error messages will result if the file was in fact not open by the FLOPEN call.

You should check the file's status when issuing subsequent calls. Use the FSTAT call to determine whether the file was opened.

## 8.0 INSTALLATION PROCEDURES

On the RX50 diskette (volume label PRO/APP DSKT BLDR V1.5) supplied, with the Tool Kit, in directory PROFMS, are two PRO/FMS-11 files:

- ☐ DEMLIB.FLB—Contains forms for the sample PRO/FMS-11 programs supplied with the Tool Kit. You will need this file on your Professional to run the sample programs.
- ☐ FMSDBG.MSG—Contains error messages for PRO/FMS-11 programs built against FDVDBG. You will need this file on your Professional to run debug versions of your programs.

To copy the files from the diskette to SYSDISK:[001,002] on your Professional, follow these steps:

1. Insert the diskette in a diskette drive slot.
2. Go to the Main Menu. Select **File Services.** Press DO.
3. The File Services menu will appear. Select **Copy File** and press DO.
4. Press ADDTNL OPTIONS.
5. The Additional Options menu will appear. Select **Choose a different directory/volume** and press DO.
6. The Directory Selection menu will appear. Select the directory **PROFMS** from the volume **Toolkit** and press DO.
7. The File Selection menu will appear. Select the file from the list of files, either DEMLIB.FLB or FMSDBG.MSG. Press DO.
8. The **Name a File Form** will appear. Press ADDTNL OPTIONS.
9. The Additional Options menu will reappear. Select **Choose a different directory/volume**. Press DO.
10. The Directory Selection menu will appear. Press ADDTNL OPTIONS.
11. The Additional Options menu will appear. Select **Choose a System Directory**. Press DO.
12. The Directory Selection menu will appear. Select **001002**. Press DO. The **Name a File Form** will reappear. Type in the filename of the file you selected, for example, DEMLIB.
13. When the file has been copied to the disk directory 001002, the File Services menu will reappear.
14. To install the other file, repeat this procedure from step 3. To display the Main Menu, press MAIN SCREEN.

## 9.0 SAMPLE PRO/FMS-11 PROGRAMS

A sample PRO/FMS-11 program in each of the Tool Kit languages is included with the Tool Kit. After the Tool Kit has been installed on your host system, you can find these program in directory LB:[1,5]. Copy the files to your own area if you wish. You'll find the accompanying forms in the file DEMLIB.FLB on the diskette distributed with the Tool Kit. Use the instructions in this manual, in the section on installation procedures, to copy the file to your Professional. The rest of this section contains listings of the sample programs, with any restrictions or comments you may need.

### 9.1 Tool Kit BASIC-PLUS-2

```
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:       Ducharme
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 illustrating 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$BUF:3410       ;HELP
311 REM EXTSCT=MS$BUF:3100       ;MESSAGE
```

```
312 REM EXTSCT=MM$BUF:0           ;MUTLI-CHOICE MENU
313 REM EXTSCT=FL$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 command 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
```

## 9.2 Tool Kit COBOL-81

When calling PRO/FMS-11 from a COBOL-81 program, you can pass variables either by descriptor or by reference.

- ☐ Use the By Reference method for numeric data type parameters, such as the starting line parameter of FCLRSH or the LUNs for the FLCHAN call.
- ☐ Use the By Descriptor method for character type parameters.

The method used will determine which interface you specifiy in your .ODL file. See the section of this manual on editing the descriptor file for details. The following sample programs demonstrate each method.

**9.2.1 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:         MEGAN
*
*       DATE:           1-APRIL-79
*
*       MODIFIED:       DUCHARME  -- To run on the Professional
*
*       DATE:           3-MARCH-83
*
*       COBOL demonstration program for FMS illustrating a
*       simple form-driven, data entry application.
*
```

```
*       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$BUF:3410      ;HELP
*       EXTSCT=MS$BUF:3100      ;MESSAGE
*       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.
```

**9.2.2 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 HLLCBL.

```
*
*       CBLDESDEM.CBL
*
*                   COPYRIGHT (C) 1979 BY
*       DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
*
*       MODULE:         CBLDESDEM
*
*       VERSION:        V01.00
*
*       AUTHOR:         MEGAN
*
*       DATE:           1-APRIL-79
*
*       MODIFIED:       DUCHARME  -- 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         ;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
*       @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.
*
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(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.
```

## 9.3 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:        Megan
C
C DATE:          1-APRIL-79
C
C MODIFIED       Ducharme
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 string 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
```

## 9.4 Tool Kit MACRO-11

Copy the files MACDEM.MAC, MACDEM.CMD, and MACDEM.ODL from your host system to your Professional before you run this sample program.

```
        .TITLE  MACDEM  - FMS DEMONSTRATION SUBROUTINE

;
; MACDEM.MAC
;
;
;                       COPYRIGHT (C) 1979 BY
;            DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
;
;
; MODULE:       MACDEM
;
; VERSION:      V01.00
;
; AUTHOR:       MEGAN
;
; DATE:         19-NOVEMBER-79
;
; MODIFIED:     DUCHARME        -- 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          ; SGD001 RMS related macros
        .MCALL  $STORE, $COMPARE, $CREATE     ; SGD001
        .MCALL  $CONNECT, $DISCONNECT         ; SGD001
        .MCALL  $PUT, $CLOSE, ORG$            ; SGD001

        $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  /SYSDISK:[1,2]DEMLIB/      ; ASCIZ 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$B                           ; SGD001 Allocate RMS FAB
        F$DEQ   2                       ; SGD001 Default file extension size
        F$ALQ   2                       ; SGD001 Allocation size for the file
        F$FOP   FB$SUP                  ; SGD001 Create new file
        F$FAC   FB$PUT                  ; SGD001 File access operations
        FAB$E                           ; SGD001 End FAB declarations

        .EVEN
RABADD:
        RAB$B                           ; SGD001 Allocate RMS RAB
        R$FAB   FABADD                  ; SGD001 Connect to FAB address
        R$RAC   RB$SEQ                  ; SGD001 Record access is sequential
        RAB$E

        .EVEN
        POOL$B                          ; SGD001 Begin Pool declarations
        P$BDB   2                       ; SGD001 Allow two buffer desc blocks
        P$FAB   1                       ; SGD001 Only one file will be open
        P$RAB   1                       ; SGD001 Need only one RAB
        P$BUF   512.                    ; SGD001 I/O Buffer space
        POOL$E

        .EVEN
RMS$LUN:
        ALUN$   OU$CHN, SY, 0           ; SGD001 Assign a Lun to the Disk
        .EVEN
        ORG$    SEQ,<CRE,PUT>           ; SGD001 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$LUN                ; SGD001 Assign the Lun 0 to the Disk
        QIOW$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              ; SGD001 Move the FAB address to R4
        MOV     #RABADD,R4              ; SGD001 Move the RAB address to R4
        $STORE  F$VAL(R0),FNA,R3        ; SGD001 Move the addr of the file name
        $STORE  F$LEN(R0),FNS,R3        ; SGD001 Move the file name size
        $STORE  #OU$CHN,LCH,R3          ; SGD001 Set the LUN in the FAB
        $CREATE R3                      ; SGD001 Create the file
        $COMPARE #SU$SUC,STS,R3         ; SGD001 Check RMS Status
        BEQ     40$                     ; SGD001 Continue if status = 1
        CALL    LEAVE                   ; SGD001 Leave on I/O error
40$:    $CONNECT R4                     ; SGD001 Connect the RAB to the FAB
        $COMPARE #SU$SUC,STS,R4         ; SGD001 Check RMS Status
        BEQ     60$                     ; SGD001 Continue if ok
        CALL    LEAVE                   ; SGD001 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                    ; SGD001 Move the addr of the FAB to R3
        $DISCONNECT R4                         ; SGD001 Disconnect the access stream
        $COMPARE #SU$SUC,STS,R4                ; SGD001 Check RMS Status
        BEQ      95$                           ; SGD001 Branch if Status OK
        CALL     LEAVE                         ; SGD001 Else I/O error
```

```
95$:    $CLOSE  R3                      ; SGD001 Close output file
        MOV     #EXTNAM,R1              ; Name of exit named data
        MOV     #ARGLST,R0              ; Get ARGLST
        MOV     F$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        ; SGD001 Move the addr data to the RAB
        $STORE  F$LEN(R0),RSZ,R4        ; SGD001 Move the len of data to RAB
        $COMPARE #0,RSZ,R4              ; SGD001 See if the data length is zero
        BEQ     10$                     ; SGD001 If not return
        $PUT    R4                      ; SGD001 Store away the string of data
        $COMPARE #SU$SUC,STS,R4         ; SGD001 Check the RMS Status
        BEQ     10$                     ; SGD001 Branch if equal
        CALL    LEAVE                   ; SGD001 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
```

## 9.5 Tool Kit PASCAL

PASCAL is not recommended for use with PRO/FMS-11. However, you can use PASCAL to access PRO/FMS-11, if you follow these restrictions:

- ☐ Source code. PRO/FMS-11 calls for PASCAL are in the Tool Kit file FMS.PAS. After the Tool Kit is installed, you can find this file in directory LB:[1,5] on your host system. 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 everything by reference. To determine the appropriate calling sequences for your application, refer to FMS.PAS and Chapter 3 of the *Tool Kit PASCAL User's Guide*.

  Note: PASCAL form and field names must be padded for six spaces. Otherwise the form or field cannot be accessed by the Form Driver.

- ☐ Parameters. There are no optional paramaters for PASCAL calls to PRO/FMS-11. All paramaters must be included. For example, in FMS.PAS, FCLRSH is declared 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 will display at the starting line specified when the form was designed.

- ☐ Indexes. When using an FMS.PAS routine that allows indexes, you must specify an index for the variable. If no index was assigned when the form was created, 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 go like this:

```
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 Chapter 3 of the *PASCAL User's Guide* for details.

- ☐ Library file specification. You must terminate a library file specification with a NUL character. Without it, PRO/FMS-11 will not be able to 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 may be displayed 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.

- ☐ FLCHAN. Be sure to assign any LUN used in an FLCHAN call to LB: in your command (.CMD) file. The files will be put in the application directory [ZZAPnnnnn].

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: S. Ducharme
        Date:   27-Apr-1983
        PASCAL Demonstration program for PRO/FMS illustratin
        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:RO
        TASK=PASDEM
        UNITS   = 20
        ASG     = TI:5:13:15
        ASG     = SY:6:7:8:9:10:11:12
        EXTSCT  = $$HEAP:10000
        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]RMSRLX
                .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 *****
        This procedure is called to attach the terminal. The Form Driver
     needs the terminal attached. The FORTRAN routine WTQIO is called 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 *****
          This procedure is called to move character data from one variable
       to another vaiable. This is useful if the two variables are of differing
       lengths. Three paramaters are passed. The two variables and the number
       of characters that are to be copied 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
{      ***** Initilize varaibles *****      }
        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 := '      ';
{      ***** Initilize FMS Impure Area and Open Library *****      }
        WTQIO(QIO_Function,TT_LUN,TT_EFN);        {Attach the terminal          }
        FINIT(Impure_Area,Impure_Size,Status_1); {Initilize 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,
        Name_Data
        );
   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.
```

PRO/FMS-11
Documentation Supplement
Order No. AA-P103B-TK

## READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

Please cut along this line.

Do Not Tear – Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

Do Not Tear – Fold Here

Cut Along Dotted Line