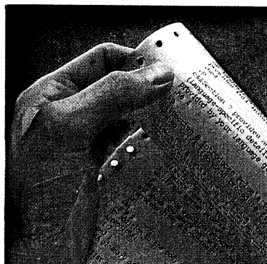
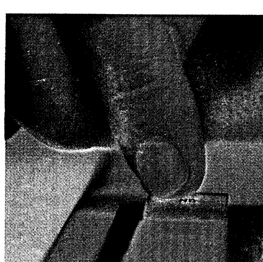
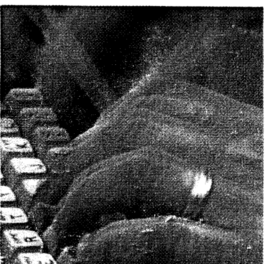
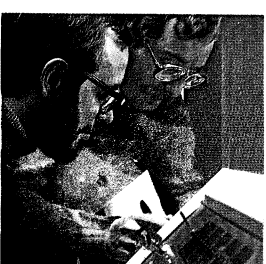
Prime Computer, Inc.
Programmer's Guide

PDR 3040-163P
FORMS
Programmer's Guide
Rev. 16.3









FORMS PROGRAMMER'S GUIDE

PDR3040

This guide documents Prime's Forms Management System (FORMS) at Master Disk Revision Level 16 (Rev. 16).

PRIME Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01760

All correspondence on suggested changes to this document should be directed to:

Maxon L. Goudy, Technical Writer
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

Acknowledgements:

We wish to thank the members of the FORMS team and also the non-team members, both customer and Prime, who contributed to and reviewed this PDR.

Copyright © 1979 by
Prime Computer, Incorporated
500 Old Connecticut Path
Framingham, Massachusetts 01701

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

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

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc. PRIMENET and THE PROGRAMMER'S COMPANION are trademarks of Prime Computer, Inc.

First Printing November 1979

Section	Title	Page
SECTION 1	INTRODUCTION TO FORMS	
	INTRODUCTION	1-1
	SCOPE OF DOCUMENT	1-1
	PURPOSE OF FORMS	1-2
	ADVANTAGES OF USING FORMS	1-2
	FORMS INTERFACES TO PRIMOS	1-3
	FORMS ADMINISTRATIVE PROCESSOR	1-6
	RELATED DOCUMENTS	1-7
SECTION 2	WRITING APPLICATION PROGRAMS FOR USE WITH FORMS	
	INTRODUCTION	2-1
	PRINCIPLES OF OPERATION	2-1
	USING FORMS	2-3
	FORMS DIRECTIVES	2-3
	PROGRAM EXAMPLES	2-4
SECTION 3	DESCRIBING DATA USED BY FORMS	
	PURPOSE OF DATA DESCRIPTION	3-1
	FORM DEFINITION	3-1
	SUMMARY OF FORM DEFINITION	3-3
	MAPPING	3-4
	FORM DESCRIPTOR PREPARATION	3-5
	TRANSLATING STREAM AND FORMAT CODING	3-7
SECTION 4	FORMS RUN-TIME DIRECTIVES REFERENCE INFORMATION	
	FUNCTION	4-1
	USAGE	4-1
	DESCRIPTION OF DIRECTIVES	4-3
	ATTRIBUTE MODIFICATION DIRECTIVES	4-8
SECTION 5	FORMS DEFINITION LANGUAGE, REFERENCE INFORMATION	
	SYNTAX OF FORMS DEFINITION LANGUAGE	5-1
	FORM DEFINITION DELIMITER STATEMENTS	5-2
	FIELD STATEMENTS WITHIN A STREAM DESCRIPTOR	5-6
	FIELD DEFINITION EXAMPLES: STREAM DESCRIPTORS	5-13
	FIELD STATEMENTS WITHIN A FORMAT DESCRIPTOR	5-14
	FIELD DEFINITION EXAMPLES: FORMAT DESCRIPTOR	5-16
	PROGRAMMING AIDS	5-16
	MACRO DEFINITION	5-17
	ITERATIVE FIELD GENERATION	5-17

RELATIVE POSITION PARAMETER 5-18
LISTING CONTROL STATEMENTS 5-19
ALTERNATE INPUT FILE ($INSERT) 5-19
FDL TRANSLATION, COMMAND FORMAT 5-2Ø
RUN-TIME MESSAGES 5-22
FDL TEMPORARY FILES 5-22
FDL COMMAND LINE EXAMPLE 5-23

SECTION 6 FORMS ADMINISTRATIVE PROCESSOR (FAP) REFERENCE INFORMATION

FUNCTION 6-1
COMMAND FORMAT 6-1
FAP COMMANDS 6-1
FAP EXAMPLES 6-9

SECTION 7 EXAMPLE FORTRAN PROGRAM

INTRODUCTION 7-1
WRITING THE PROGRAM 7-3
CREATING THE FORM DESCRIPTOR FILE 7-11
COMPILING THE APPLICATION PROGRAM 7-18
TRANSLATING THE FDL SOURCE 7-18
INSTALLING FORM DESCRIPTOR IN FORMS CATALOG 7-18
LOADING THE APPLICATION PROGRAM 7-18
RUNNING TEH PROGRAM 7-19

SECTION 8 EXAMPLE COBOL PROGRAM

INTRODUCTION 8-1
WRITING THE PROGRAM 8-3
CREATING THE FORM DESCRIPTOR FILE 8-9
COMPILING THE APPLICATION PROGRAM 8-13
TRANSLATING THE FDL SOURCE 8-13
INSTALLING FORM DESCRIPTOR IN FORMS LIBRARY 8-14
LOADING THE APPLICAITON PROGRAM 8-14
MIDAS FILE TEMPLATE 8-15
RUNNING THE PROGRAM 8-16

APPENDIX A INSTALLATION A-1

DIRECTORY INFORMATION A-1
INSTALLING A NEW VERSION OF FORMS A-1
UPGRADING A CURRENT INSTALLATION A-2
REBUILDING FORMS A-3

APPENDIX B	DEVICE I/O	B-1
	DEVICE INPUT/OUTPUT SYSTEM	B-1
	IOCS INTERLUDE	B-1
	DEVICE I/O MECHANISM	B-1
	PRIME-SUPPLIED DEVICE DRIVERS	B-3
APPENDIX C	USER-WRITTEN DEVICE DRIVERS	C-1
	INTRODUCTION	C-1
	INSTALLING THE DEVICE DRIVER	C-6
APPENDIX D	TROUBLE SHOOTING	D-1
APPENDIX E	SAMPLE FORTRAN PROGRAM	E-1
APPENDIX F	FORM DESCRIPTOR FOR FORTRAN PROGRAM EXAMPLE	F-1
APPENDIX G	SAMPLE COBOL PROGRAM LISTING	G-1
APPENDIX H	FORM DESCRIPTOR FOR COBOL PROGRAM EXAMPLE	H-1
APPENDIX I	ADVANCED USE OF FORMS	I-1
APPENDIX J	ERROR MESSAGES	J-1
	ERROR MESSAGE FORMAT	J-1
	FDL ERROR MESSAGES	J-1
INDEX		X-1

SECTION 1

INTRODUCTION TO FORMS

INTRODUCTION

The Prime Forms Management System (FORMS) provides a convenient and natural method of defining a form with a language designed for such a purpose. Defined forms may then be read or written by any application program that is capable of using Prime's Input-Output Control System (IOCS). Application programs communicate with FORMS through input/output statements native to the host language. (The host language is the language in which the application program source was written.) Programs that currently run in an interactive mode may easily be converted to use FORMS.

SCOPE OF DOCUMENT

This document is divided into four parts.

The first part, Sections 1 through 3, contains TUTORIAL INFORMATION. These sections are intended to give the user a brief introduction to forms and quickly show the user how to use FORMS.

The second portion of this document, Sections 4 through 6, contains REFERENCE INFORMATION. Some of the material discussed in Sections 1 - 3 is repeated, but these sections give more detail on each topic relating to FORMS. Section 4 is a detailed discussion of the FORMS run-time directives used in the coding of application programs. Section 5 describes in detail how to create a form descriptor, which describes to FORMS the complete format of a form both on the terminal display screen and in the computer system data record. Section 6 tells in more detail the features for storing and maintaining form descriptors in the FORMS catalog.

The third portion of this document, Sections 7 and 8, are of particular interest to the programmer. Section 7 describes an application program written in FORTRAN, and Section 8 discusses an example program written in COBOL.

The last part of the document is the appendices. The appendices contain information that are needed less frequently by the FORMS user, for instance "Installation". Appendices also contain information to support material presented in earlier sections, such as program listings and form descriptor listings. Advanced FORMS usage and error messages are the final appendices.

PURPOSE OF FORMS

FORMS simplifies and standardizes the transfer of data fields (or groups of data fields) between application programs and page-oriented video terminals and hard-copy devices. FORMS provides easy-to-use facilities for defining how data fields are to be displayed at or received from one or more block-mode terminal types, and FORMS uses the data definitions at run-time to automatically control the data transfer.

In addition, FORMS provides centralized administrative control and simplified maintenance. Through commands to the FORMS Administrative Processor, a user can add, delete, or change either the forms themselves, or the terminal type.

Finally, FORMS provides the user with a set of run-time directives that reside in a special subroutine library.

ADVANTAGES OF USING FORMS

Because the user-terminal screen can be formatted to resemble a source document (i.e., the form on paper), FORMS is easy to use, and allows user personnel to be trained quickly. Also, any input data errors may be corrected before they are read by the application program since visual verification of the data at the terminal, the ability to retype data fields, and the ability to highlight errors are all possible.

Some additional advantages are:

1. The user is more comfortable using a form displayed at the terminal that looks the same as the printed form.
2. The user can enter information in any order, by moving the terminal cursor to the correct data item location.
3. The user can see the information and can see what information, if any, is missing.
4. It is possible for the user to make corrections to entries in the form <u>before</u> transmitting them from the user terminal to the computer.
5. Computer processing time is minimized while updating or entering a form since FORMS "off loads" the CPU, i.e., it is not interacting with the user on a character by character basis.

FORMS INTERFACES TO PRIMOS

Prime's Forms Management System consists of three major components: the FORMS Description Language (FDL), the FORMS Administrative Processor (FAP), and the FORMS Run-Time library. These components work together to create, administrate, and run forms-oriented applications. Figure 1-1 shows a functional overview of FORMS and its relation to some PRIMOS system components. FORMS can be applied quickly in simple logical steps since each component is independent.

Figure 1-1 illustrates the various steps that go into the creation and use of a FORMS program.

Part A of the figure shows the development of the applications program. First, the source code in created, using Prime's text editor (ED). The source code is then translated by the appropriate compiler or assembler (FTN, COBOL, RPG, PMA). The object code is then loaded using SEG or LOAD, and the applications program is ready to run. (See the appropriate language user's guide for details).

The program thus created contains FORMS directives (for example, in FORTRAN, the directives are in FORMAT statements). Section 2 of this guide explains how to write applications programs that include FORMS directives. Section 7 illustrates this process with a sample FORTRAN program; section 8 illustrates it with a sample COBOL program.

Part B of Figure 1-1 shows the dual nature of I/O in the executing program. Standard I/O statements interface with the terminal through Prime's Input/Output Control System (IOCS, see PRIMOS Subroutines Reference Guide). FORMS I/O directives interface through the FORMS run-time package.

Part C of Figure 1-1 shows how the run-time package receives information. Notice how closely the user's work here parallels the user's creation of the applications program.

- The editor is used to create the form definition source: i.e., the STREAM and FORMAT descriptors. Section 3 of this guide explains how to create these descriptors.

- The FDL translator is invoked to translate the source code into binary. Use of FDL is explained in Section 5.

- The Forms Administrative Processor (FAP) loads the object code into the FORMS directory, where it is available for run-time use. Section 6 of this guide explains how to use FAP.

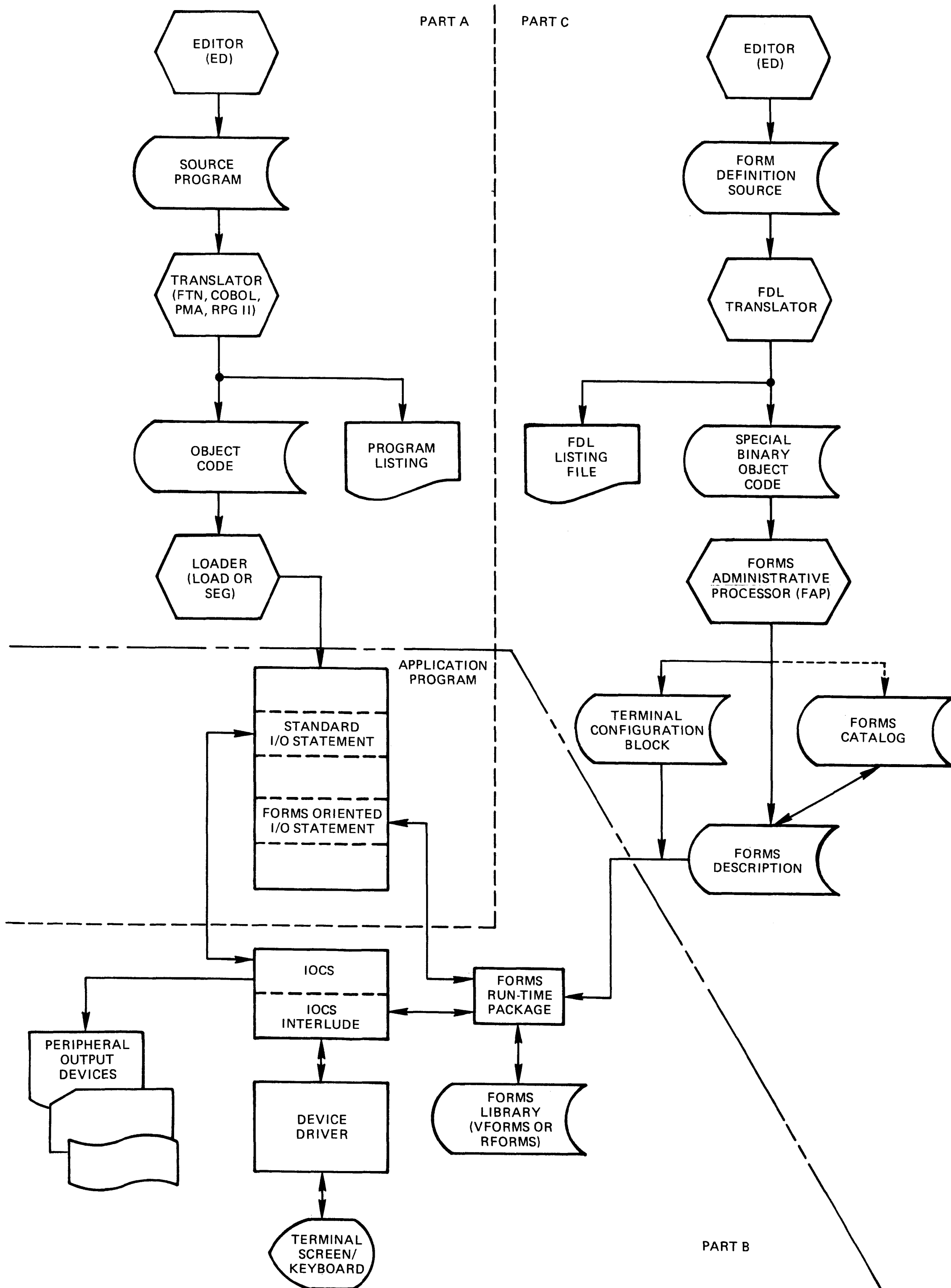


Figure 1-1.
Functional Overview, FORMS and PRIMOS

Programming Language Interface

The application program source may be written in COBOL, FORTRAN, RPG-II, PMA, or any language capable of interaction with Prime's Input-Output Control System (IOCS). For further information about IOCS, refer to the Reference Guide, PRIMOS Subroutines. For information on programming in the languages mentioned, refer to the appropriate language reference guide. Application programs communicate with FORMS by means of input/output statements specified within the format of the source language (e.g., FORTRAN formatted READ and WRITE statements).

Terminal Software Interface

FORMS provides a means for displaying information on the screen. The FORMS display on the screen shows what fields are available for the user to type-in data to be transmitted to the application program. The screen handling allows the user to validate data because nothing is transmitted to the application program until the transmit key is pressed at the terminal. Thus, the user can enter data, visually verify it, move the terminal cursor to any incorrect entry and enter the correct data, and transmit each line of data.

Operating System Interface

FORMS operates under the control of PRIMOS, the Prime operating system.

Additionally, because the FORMS interface to application programs is through standard read/write statements, existing batch-oriented programs can be readily adapted to run using local and remote terminals without major re-programming.

FORMS keeps application programs, the forms they use, and terminals they use separated until run-time so that changes can be made in one area without necessarily affecting the other two. With this flexibility, terminal types may be changed; the way a form is organized at a particular terminal may be changed; or old forms may be described for new terminals. These changes, and many more, are all possible without affecting operational programs.

FORMS ADMINISTRATIVE PROCESSOR

Purpose

The Forms Administrative Processor (FAP) allows the user to maintain the FORMS catalog. This catalog contains the object files created by FDL. FAP also allows control of the terminal types and lines that are associated with a form.

FORMS Definition Catalog

The FORMS definition catalog is a segment directory that contains the binary representations, generated by FDL, of all the STREAM and FORMAT descriptors (see Section 3) available within the system configuration. After a form definition is translated by FDL, it is entered in the FORMS definition catalog with the FAP (Forms Administrative Processor) command. Refer to Section 6 for details of the FAP command. The FORMS catalog is in the FORMS system UFD, FORMS*.

FAP Functions

FAP performs the following:

- CREATE the FORMS catalog UFD and the segment directory associated with the specified form.
- ADD binary files translated by FDL to the FORMS directory.
- REPLACE binary files in the FORMS directory.
- PURGE (delete) binary files from the FORMS directory.
- LIST the binary files in the FORMS directory.
- Add, replace and remove user terminal types on the Terminal Configuration Block.
- GENERATE $INSERT files for the run-time device subroutine.

RELATED DOCUMENTS

The following Prime documents contain additional supporting information for the FORMS user.

Language Manuals

Document No.	Title
PDR3031	RPG II Programmers Guide
PDR3056	COBOL Programmers Guide
FDR3057	FORTRAN Programmers Guide
FDR3104	New User's Guide to Editor and Runoff

Operating System and Utilities

Document No.	Title
PDR3061	Reference Guide, MIDAS
FDR3108	Reference Guide, PRIMOS Commands
PDR3621	Reference Guide, Subroutines

SECTION 2

WRITING APPLICATION PROGRAMS FOR USE WITH FORMS

INTRODUCTION

This section gives an overview of how to write application programs for use with FORMS. A more detailed discussion of this task is provided in the programming examples in Sections 7 and 8.

Figure 2-1 shows the flow of instructions and data in an application using FORMS. The user must control that flow of data by special I/O statements that use FORMS run-time directives.

This section discusses how to write I/O statements using the FORMS run-time directives. I/O statements are the only interface to FORMS in the application program. Sections 3 and 6 describe how to write a form descriptor and install it in the FORMS catalog (a UFD named FORMS*).

PRINCIPLES OF OPERATION

Programs that are written in standard languages such as FORTRAN or COBOL can interact with formatted (sometimes called "block mode") terminals in either one of two ways. One solution is the user can supply, in the application program, all of the control character sequences to the terminal that are needed to output each field (data area). This usually requires several bytes of cursor positioning information, plus field identifier and attribute bytes, plus the field data, and the field terminator byte. The application program can interpret the character input stream from the terminal, distinguish input data from control data, and process each datum accordingly. This is an awkward and somewhat difficult task to program in either FORTRAN or COBOL.

Although there are short cuts (a subroutine package for example), the application programmer becomes more concerned about device characteristics than with the application to be accomplished. Furthermore, once the application program is written, changing the format of the terminal screen definition described within the application program is difficult. The application program must often be rewritten if another type of terminal device requires support. In summary, program maintenance using this approach is time-consuming and costly.

An alternative solution to terminal I/O being controlled by the application program is the Prime FORMS management system. FORMS allows the user to describe data formats in a forms description language (refer to Section 3). This definition language is completely separate from the application program. The form definition serves as an interface between the application program and the page-oriented

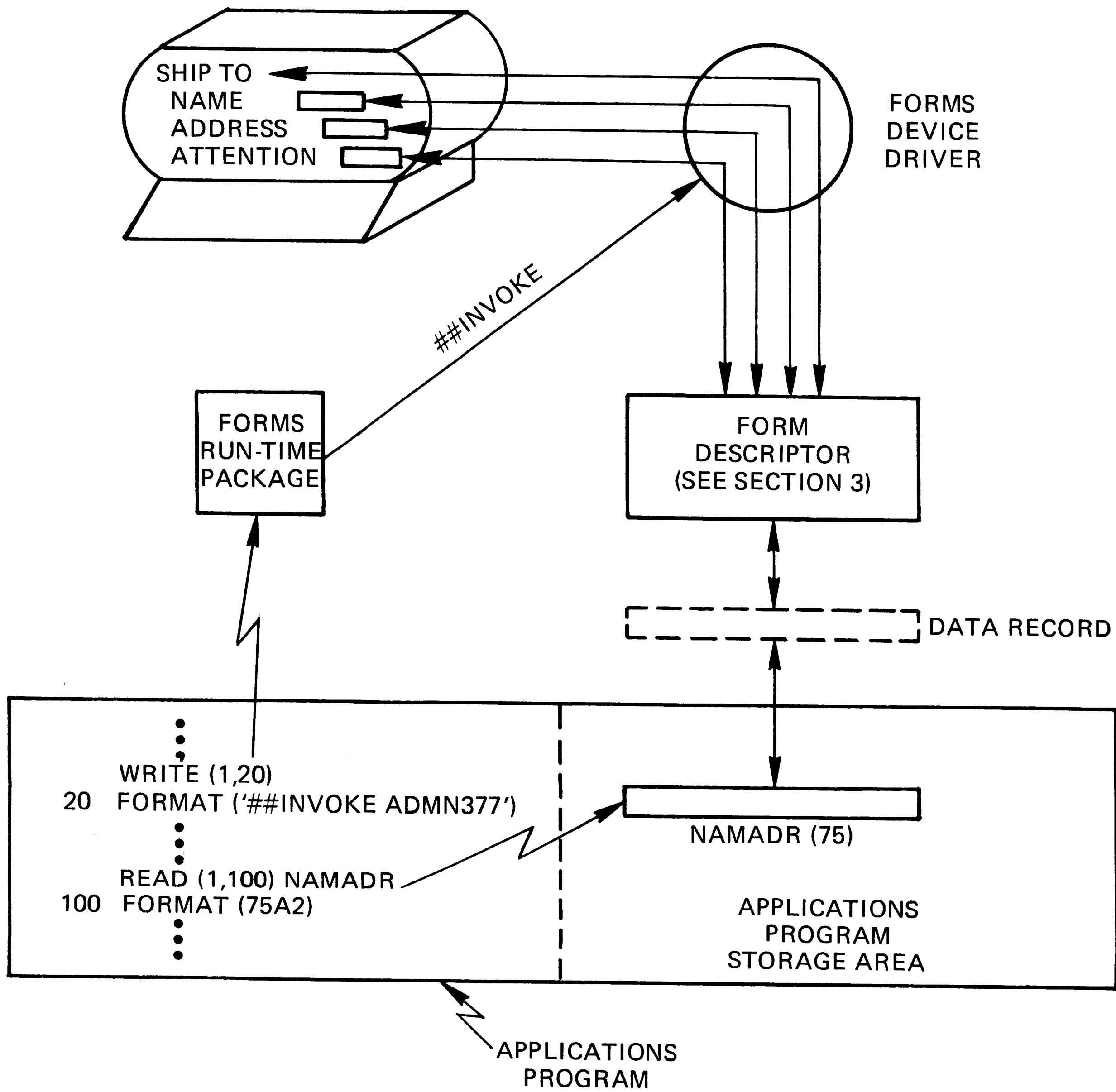


Figure 2-1. Flow of Instructions and Data between Terminal Screen, FORMS Descriptor, and Applications Program Data Area.

terminal device in use. The forms definition describes each data field transferred (input) to or (output) from the application program by its position in the input or output data record and further relates this description and position to the field's position on the terminal screen. FORMS also relates other information such as the field's length, display attributes (blink, reverse-video, write protected, etc), justification, validation - if any, etc. Data is transferred between the application program and the terminal using standard input and output statements (for example, READ and WRITE).

USING FORMS

FORMS consists of three components. A form definition language translator (FDL) translates source form definitions into a usable form. A catalog maintenance tool, the Forms Administrative Processor (FAP), is used to update a system-wide FORMS directory that contains all form definitions available for use by application programs. Finally, the FORMS Run-time System Library is a collection of subroutines that are invoked by FORMS directives and interact with the application programs and the terminals to provide I/O handling at execution time.

FORMS is device independent. The user may define a form to be displayed on any terminal and/or system (spooled) line printer. (FORMS works with any page oriented device, hard or soft copy.) Multiple terminal types may simultaneously run the same application program, since physical device selection is deferred until execution time.

To use FORMS, the user must have a terminal that has field write-enable/protect, absolute cursor positioning, and block-mode transmission capabilities (refer to Appendix B). FORMS currently supports the VISTAR3 and OWL1200 devices and system line printer. If the user wishes to use a nonstandard terminal, only a device driver subroutine need be written following the guidelines set forth in Appendix B.

FORMS DIRECTIVES

The only difference between a "normal" program and a FORMS application program is the inclusion of one or more FORMS directives. FORMS directives consist of keywords preceded by a double pound sign, for example, ##INVOKE. These directives are embedded in standard language constructs. In FORTRAN, the directives are enclosed in quoted ASCII strings in FORMAT statements. For example,

```
      WRITE (6, 100)
      .
      .
      .
100    FORMAT ('##INVOKE SCREEN1')
```

As demonstrated in the previous example, FORMS operates on a single file unit (default is 6) for screen device I/O. All I/O to the device takes place on that unit.

In the FORMAT statement example, the FORMS directive causes a character string to be passed to the I/O processor. This string is intercepted for special handling (the double pound sign is the token that prompts this interpretation). The ##INVOKE directive, when intercepted, causes a call to be made to the FORMS run-time subroutine library, and the function of INVOKE is performed, namely the form descriptor named SCREEN1 is opened and written to the user terminal. Thereafter, every transaction of file unit 6 is controlled by the FORMS library file known as SCREEN1. Sections 3 and 6 discuss more about forms descriptions and how to create and install them.

Run-time directives are the link between the application program, the catalogued form descriptions, the terminal configuration, and the terminal device drivers. For example, one directive selects a form for a particular application (##INVOKE). Directives pass all data fields a FIELD or FORMAT at a time, check validation status of input fields, clear displayed data fields, and change data field attributes during program execution. The available run-time directives and their functions are discussed briefly in Table 2-1. For a detailed discussion of the run-time package, refer to section 4.

PROGRAM EXAMPLES

Section 7 gives an example program written in FORTRAN and Section 8 gives an example program written in COBOL.

Table 2-1. Summary of FORMS Directives

Directive	Description
##BLINK/##NOBLINK	Blinks a designated field (##BLINK) or turns off blinking (##NOBLINK) feature.
##CLEAR	Clears unprotected (i.e., variable) data field displayed on the user terminal. (##CLEAR ALL clears all unprotected fields.)
##DISPLAY/##NODISPLAY	Displays field when form is output (##DISPLAY) or does not display field (##NODISPLAY).
##FKEYS	Enables (##FKEYS ON) or disables (##FKEYS OFF) user function keys input at the terminal.
##FORCEREAD	Forces FORMS to wait for and process user input at the terminal.
##INVOKE	Defines the form definition to be used (needed at least once in every applications program).
##PRINT	Allows output of current form and sends data to a printer.
##POSITION	Specifies field to which the cursor at the terminal will be positioned on the next read operation.
##PROTECT/##NOPROTECT	Write-protects a designated field (##PROTECT) or removes protection (##NOPROTECT).
##RELEASE	Specifies that the current form definition is no longer to be used.
##RVIDEO/##NVIDEO	Displays field in reverse video (##RVIDEO) or normal video (##NVIDEO).
##SUBSTREAM	Defines a substream to be processed by the next read or write statement.
##VALIDATE	Causes the return of validation status for all input data.

SECTION 3

DESCRIBING DATA USED BY FORMS

PURPOSE OF DATA DESCRIPTION

Video terminals have a great potential for simple, powerful data entry and retrieval. Such features as page mode, cursor control, reverse video, and blinking can improve the speed and convenience of data terminal use. However, use of these features places a burden on the application program.

The Prime FORMS software package makes a terminal with page mode and cursor control look like any ASCII input-output device (as far as the application program is concerned.) Terminal input-output data is exchanged a record at a time by the normal I/O rules of the language in which the application program is written. The input and output formats of the data transferred to and from the terminal and the computer system must be indicated to FORMS (and the application program). With FORMS, the templates of the form and the data do not have to be specified internally in the application program. Instead, form and data descriptions are specified externally by the user and translated by the Forms Definition Language (FDL) command.

FORM DEFINITION

A form definition describes user and terminal data formats. The form definition is divided into two parts. The first part describes the input/output data record. This descriptor gives the location of each data item in the record as well as some information about each data item in the record. The second part of the form definition describes how each of the data items (or fields) are displayed at the terminal. A typical form definition is discussed in the following paragraphs.

Consider the design of a simple inquiry program that uses FORMS to display entries at the terminal from a keyed index file. Program operations consist of entering an employee-id number at the terminal. Using the given employee-id, the program performs a file look-up and displays the information to which the employee-id pertains. If the employee-id entered is zero or spaces, the program will exit to PRIMOS command level.

Columns	item name	data type
1-4	employee-id	numeric
5-34	employee-name	alphabetic
35-64	street-address	alphanumeric
65-84	city	alphabetic
85-86	state	alphabetic
87-91	zip-code	numeric
92-103	phone	numeric / special

Graphically, the above listed data would appear in the data record definition as follows:

1—4	5---34	35-----64	65--84	85---86	87-91	92--103
id	name	address	city	state	zip	phone

This information illustrates the first part of the form definition. This part is known as the data stream descriptor. The data stream descriptor contains STREAM descriptor fields that describe each item in the user's input/output data record(s). The data stream descriptor must include the length of each item, and either implicitly or explicitly include the position of each item within the data record (i.e., the starting character position must be known or must be capable of being determined). The field descriptor(s) may optionally include a justify specification. FORMS provides for left-justification, right-justification, or centering. FORMS also provides a zero-fill or space-fill option when entering data with justification. FORMS validates input data under a specified validation mask, or series of masks, and allows the user to correct the data if it is incorrect.

The next task to be accomplished in the example inquiry program, previously discussed, is to design the FORMAT of the data displayed at the terminal. The display size (number of columns and lines available) must be taken into consideration. Attributes like write-enabled, blink, noblink, reverse video, or normal video that may be applied to each data field must also be considered. Finally, the length of each field displayed at the terminal must be specified. This parameter may differ from the length of the corresponding field in the input or output data record. In addition, when considering the length of fields displayed at the terminal, any field proximity restrictions imposed by the device must be taken into account, for example, some terminals require fields be separated by one or more blanks.

While assigning physical device positions and attributes for each data field in the data STREAM descriptor, the user may also specify titles (i.e., literal data) in the form definition which is displayed at the terminal along with the application program data. This literal data usually describes or identifies data fields that follow.

The information used to construct the second part of the form definition is known as the device format descriptor. An example of this type of information is shown by the following table:

line	column	content	length	attributes
2	2	'EMPLOYEE ID'	11	write-protected
2	20	employee-id	4	write-enabled
4	2	'NAME'	4	write-protected
4	20	employee-name	30	write-protected
6	2	'ADDRESS'	3	write-protected
6	20	street-address	30	write-protected
7	20	city	20	write-protected
7	45	state	2	write-protected
7	50	zip	3	write-protected
9	2	'HOME PHONE'	12	write-protected

According to the information in the foregoing list, when the form is written to the terminal at application program run-time, the information should appear as follows:

```
       ....*....1....*....2....*....3....*....4....*....5....*....
      ------------------------------------------------------------
  1 |
  2 |  EMPLOYEE ID        ____
  3 |
  4 |  NAME               ******************************
  5 |
  6 |  ADDRESS            ******************************
  7 |                     ********************    **   *****
  8 |
  9 |  HOME PHONE         ************
```

The line and column markers in the illustration have been provided for ease of position identification. They do not appear when the form is displayed at the terminal. The underline characters represent write-enabled data. Write-enabled data may be modified at the terminal by the user by typing new information in the appropriate field. The asterisks in the illustration represent write-protected data, which may not be modified by the user at the terminal.

SUMMARY OF FORM DEFINITION

This section gives information on how a form is specified and how to use the FDL command to translate the FDL source input. With FORMS, the data fields within a form are stored as a stream and must be converted to some format to be displayed at the user's terminal device.

A format indicates the locations on a terminal screen where literal text and variable data are to appear.

A stream indicates the mapping of the fields passed to and from the data record (in the files maintained by the application program) and those fields displayed on the terminal screen.

Together, the STREAM and the FORMAT are called the form definition.

MAPPING

Mapping establishes a correlation between a STREAM descriptor field and a FORMAT descriptor field. Mapping binds the record position and item (field) length information etc. contained in the STREAM field to the terminal device position and display attribute information contained in the FORMAT field.

Fields in the data STREAM descriptor are mapped by name to corresponding fields in the FORMAT descriptor. These field names need not match the data names used within the application program, but common sense and good programming practice dictate that the user should, where practical, keep names as closely matched as possible. For example, a data item called EMPLOYEE-NAME in a COBOL program might be represented by a FIELD called EMPLNAME in a form definition. There is an eight-character maximum for names within FORMS.

STREAM descriptor fields that are mapped (some are not, refer to Section 5) contain the name of the FORMAT descriptor field to which they map in the body of the FIELD definition. Therefore, FORMAT descriptor FIELDs are mapped to STREAM descriptor FIELDs and are also assigned a name by the associated STREAM descriptor FIELD. Two fields are bound when the name specified in the STREAM descriptor FIELD and the fieldname specified in the FORMAT descriptor FIELD are identical. A bound field contains all available information regarding the data therein: i.e., its record and terminal device position and length, justification, validation, and display attributes.

Because the two parts of the form definition are described separately in The Form Definition Language (FDL), they must be in some way related to each other. Since STREAM fields are mapped to FORMAT fields, so are STREAM descriptors mapped to FORMAT descriptors. Each data STREAM descriptor has an associated unique name. When the application program uses a form definition, it is identified by this name. FORMAT descriptors that correspond to a particular STREAM descriptor are assigned an identical name.

Refer to the sample STREAM and FORMAT previously discussed. The STREAM descriptor name must be the same as the FORMAT descriptor name. For example, the STREAM descriptor FIELD describing the employee-id must map to the FORMAT descriptor FIELD containing the employee-id. Likewise, the STREAM descriptor FIELD describing the employee-name must map to the FORMAT descriptor FIELD describing the employee-name, and so on.

FORM DESCRIPTOR PREPARATION

The STREAM and FORMAT are coded in Forms Definition Language. This is the source input to the FDL translator. The source input may be prepared using the PRIMOS text editor (ED) and may be placed into a file. For information on the editor (ED), refer to the New User's Guide to Editor and Runoff. The STREAM and FORMAT descriptors are translated by the FDL translator. When translated, the object output is placed in the FORMS catalog using the FAP command. (For further information, refer to Section 6.)

Coding a FORMAT

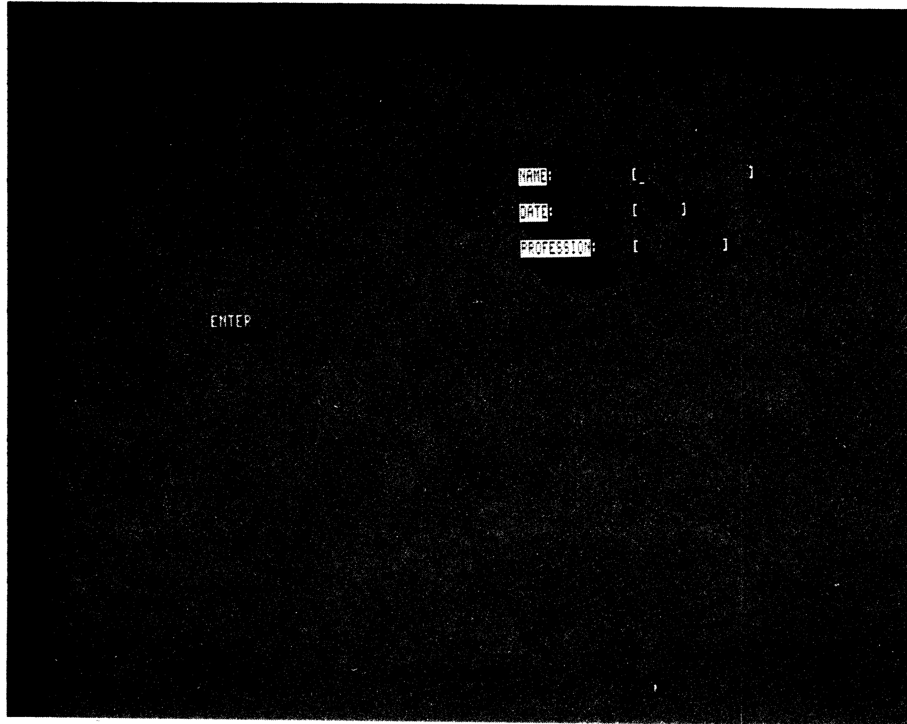
Figure 3-1 shows a simple screen layout, amd the associated FORMAT coding needed to define the terminal screen layout that is illustrated.

From the example shown in Figure 3-1, it is possible to derive some guidelines of how a FORMAT is defined.

- The input line is generally free-format.
- FORMAT descriptors and FIELD names must start in column (character position) 1 and be followed by a space.
- FIELD and FORMAT names must be from 1 to 8 alphanumeric characters (A-Z, 0-9).
- FIELD statements may start anywhere after column 1 and may occupy columns 2-72.
- Columns 73-80 are ignored by FDL.
- Items in an FDL source input line may be separated by a space or a comma.

Figure 3-1 also shows some of the fields specified as NOPROTECT. It is a useful practice when specifying a field having the NOPROTECT attribute, to allow two (2) screen positions either side of the NOPROTECTed field for possible insertion of control characters.

The boundaries of a FORMAT descriptor are a FORMAT statement and an END FORMAT statement. Within the FORMAT definition, there must be at least one DEVICE statement and an associated END DEVICE statement. Within the definition of a DEVICE, there may be as many FIELD statements as are necessary to define the form. FIELD statements define both literal and variable (NOPROTECTed) fields. Optionally, REPEAT and END REPEAT statements may be included as desired within the boundaries of a DEVICE definition to define similar fields which are repeated a number of times on a form, for example, part numbers and part descriptions on a parts list.




```
INFILE FORMAT
        DEVICE OWL1200
        FIELD 'NAME' POSITION (21,8), REVERSE VIDEO
        FIELD ':    ' POSITION (25,8)
NAM     FIELD LENGTH 16, POSITION (38,8), NOPROTECT
        FIELD 'DATE' POSITION (21,10), REVERSE VIDEO
        FIELD ':    ' POSITION (25,10)
DAT     FIELD LENGTH 6, POSITION (38,18), NOPROTECT
        FIELD 'PROFESSION' POSITION (21,12), REVERSE VIDEO
        FIELD ':    ' POSITION (31,12)
PROF    FIELD LENGTH 12, POSITION (38,12), NOPROTECT
        END DEVICE
        END FORMAT
```

Figure 3-1. Sample FORMAT Code and Screen Display.

Coding a STREAM

Figure 3-2 shows a simple data record layout and the associated STREAM coding needed to define the data record used by the application program to store the form within the record. From the example shown in Figure 3-2, it is possible to derive some guidelines about how a STREAM is coded.

There is a slight alteration on the appearance of a FIELD statement in a STREAM descriptor, compared to a FIELD in a FORMAT descriptor. Examination of Figures 3-1 and 3-2 reveals the relationship that the FIELD names used in the FORMAT must be the same as the identifiers in the FIELD statements in the STREAM specification. Also, the FORMAT name and the STREAM name are the same. This parallelism of names is called mapping, and has been discussed in previous paragraphs. The reason for mapping is to associate the data items in fields on the screen with their counterparts in the fields stored in the data record within the computer system.

The guidelines for coding a STREAM are essentially the same as for coding a FORMAT. The same rules hold for STREAM descriptors as for FORMAT descriptors.

A STREAM descriptor begins with a STREAM statement, and ends with an END STREAM statement. The only other kind of statements needed within the body of a simple STREAM are FIELD statements. However, these FIELD statements need not have names - though they can - but must have an identifier within the FIELD statement that maps to the same FIELD name in an associated FORMAT descriptor. Optionally, REPEAT and END REPEAT statements may be used as needed to lay out the description of fields in the data record that are repeated a number of times.

When coded, the STREAM is translated by FDL. The next portion of this section discusses translation.

TRANSLATING STREAM AND FORMAT CODING

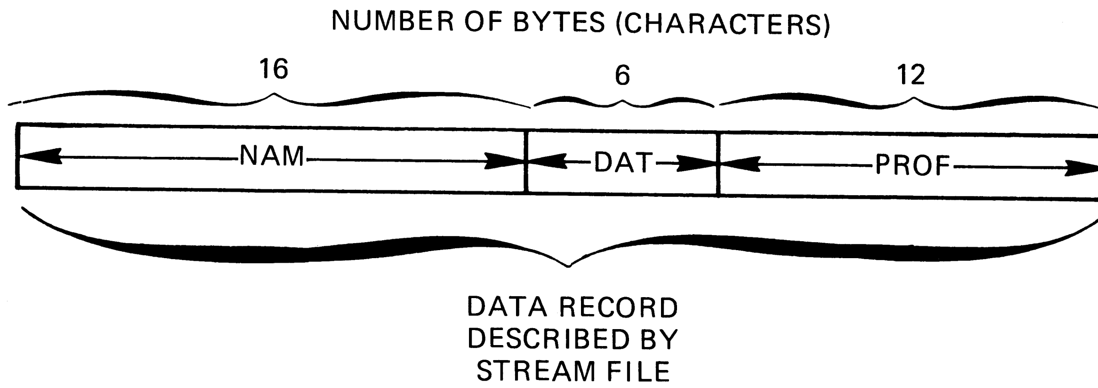
To translate a source form definition, the command is:

```
FDL filename
```

filename is the name of the file that contains the source text of the FORMAT and STREAM produced by the user. For example:

```
OK, FDL DATAS1
GO
0000 ERRORS (FDL, REV 16 - 16-FEB-79)
0000 ERRORS (FDL, REV 16 - 16-FEB-79)

OK,
```




```
INFILE     STREAM
           FIELD NAM LENGTH 16, NOPROTECT
           FIELD DAT LENGTH 6, NOPROTECT
           FIELD PROF LENGTH 12, NOPROTECT
           END STREAM
```

Figure 3-2. Sample Data Record Layout and STREAM Coding.

The FDL translation produces a binary-form object file and a listing file named B_filename and L_filename, respectively. The above example produces an object file named B_DATAS1 and a listing named L_DATAS1.

FDL Output Listing

The listing file for a FORMAT translation provides a listing of the statements within the FORMAT descriptor and a diagram of the FORMAT that is translated.

The listing file for a STREAM descriptor also provides a listing of the statements within the STREAM descriptor. This listing always specifies both an input STREAM descriptor listing and an output STREAM descriptor listing. If not otherwise specified, all STREAM fields are considered to be INPUT/OUTPUT.

Examples of the output listing from FDL will be found in Appendix F and Appendix H.

Table 3-1 lists the options available for use with the FDL command.

Table 3-1. FDL OPTIONS

Option	Meaning
-OBJLIST	Produce listing with translated code represented in octal.
-MACLIST	Lists macro expansions.
-ERRLIST	List lines containing ERRORS only. If errors are present, this option overrides all others.
-ERRTERM	Display (or print) lines containing errors at the user terminal.
-REPLIST	Generate expanded REPEAT block listings.
-NOMACLIST	Suppress expand macro listings.
-NOERRTERM	Suppress error output to terminal.
-NOREPLIST	Suppress expanded REPEAT block listings.

SECTION 4

FORMS RUN-TIME DIRECTIVES
REFERENCE INFORMATION

FUNCTION

The FORMS run-time package provides a series of directives to perform all form definition lookup, directive processing, data manipulation, and device input/output (refer to Figure 4-1).

The directives available in the FORMS run-time package are listed below. The following directives are generally concerned with form and data input-output:

- ##CLEAR
- ##FKEYS
- ##FORCEREAD
- ##INVOKE
- ##POSITION
- ##PRINT
- ##RELEASE
- ##SUBSTREAM
- ##VALIDATE

In addition, there are a number of FORMS directives that control the manner in which fields are displayed at the terminal. These are:

- ##BLINK/##NOBLINK
- ##DISPLAY/##NODISPLAY
- ##PROTECT/##NOPROTECT
- ##RVIDEO/##NVIDEO

USAGE

The application program passes to FORMS any statement that contains a forms directive. A FORMS directive is written with two preceding hash marks (##) in order to be identified by the program as a FORMS directive. These directive are extensions of the source programming languages in which the application program is written.

For example, to invoke form FD4190 on the terminal and protect fields FIELDA, FIELDB, and FIELDC, a FORTRAN application program would execute the following directives:

```
       WRITE (1,400)
 400   FORMAT ('##INVOKE FD4190'/
      + '##PROTECT FIELDA, FIELDB, FIELDC')
```

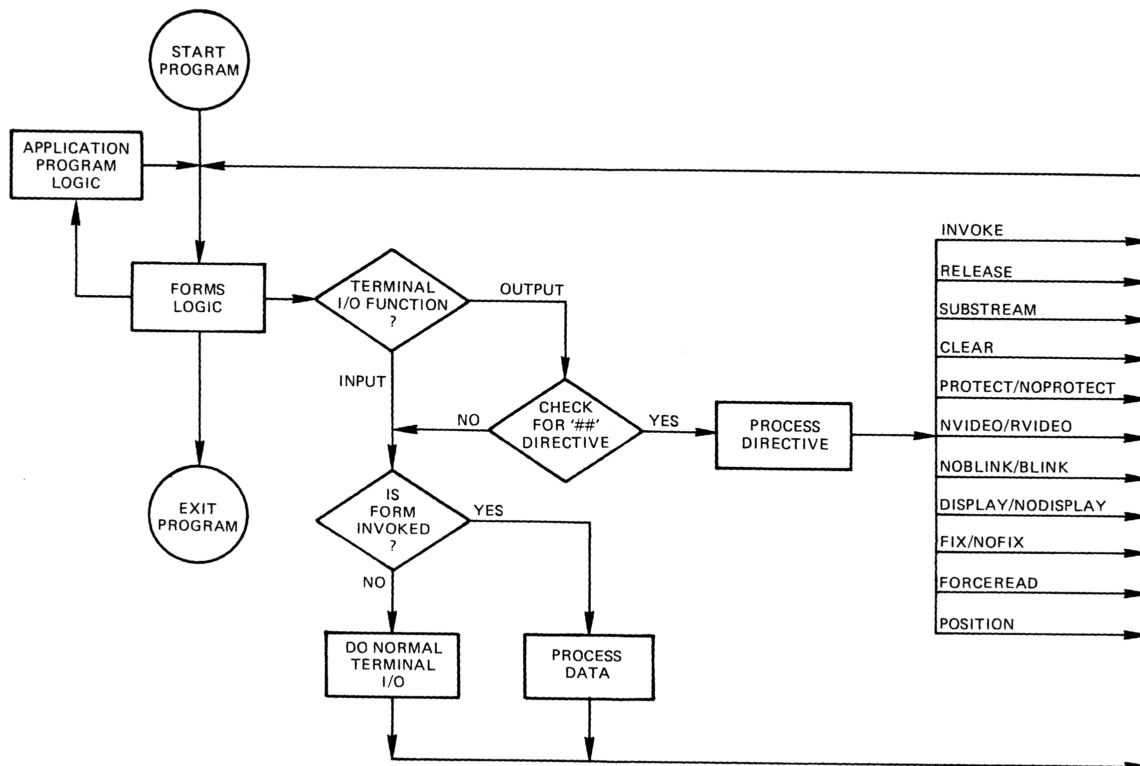


Figure 4-1. FORMS Run-time Package Functional Relationships.

DESCRIPTION OF DIRECTIVES

The following paragraphs describe all directives available in the FORMS run-time package. For clarity, they are depicted in upper-case characters. Lower-case characters in FORMS directives are mapped to upper-case.

▶ ##CLEAR [ALL]

The ##CLEAR directive clears all unprotected data displayed on the user terminal. It also causes all data items marked as NOPROTECTed and displayed in the input/output list to be reset to spaces. This is a fast and convenient method to erase all user-input data. Alternately, spaces may be written to all unprotected fields on the form. If the ##CLEAR directive is followed by the word ALL, the entire display is erased. This should be done only prior to issuing a ##RELEASE directive.

##CLEAR Examples

```
        WRITE (1,4000)
400     FORMAT ('##CLEAR')
              .
              .
C--- CLEAN UP BEFORE EXITING TO COMMAND LEVEL.
C
        WRITE (1,5020)
5020    FORMAT ('##CLEAR ALL'/ '##RELEASE')
```

▶ ##FKEYS {OFF | ON}

The ##FKEYS directive enables, or disables, user function key input from the terminal. If ##FKEYS is followed by the parameter OFF, the function keys at the terminal keyboard are disabled; if followed by the parameter ON, function keys are enabled.

When function keys are disabled, they have no effect if entered. The standard response is to place a warning message on the terminal and wait for the user to press the transmit key. When a form is invoked, an implicit ##FKEYS OFF directive occurs.

When function keys are enabled, a two-digit code is automatically appended to each input record following the right-most FIELD defined in the STREAM or SUBSTREAM. This field contains the number of the function key that was pressed when the data was transmitted from the device. If the normal transmit key was pressed, this field contains 00.

If multiple substreams are described in the form definition, the function key number is appended to each.

It is the user's responsibility to ensure that the two character positions required for the function key field are available at the end of each input record. If these two character positions are not available, the function key number is not returned.

The application program may define each function key to perform some special escape function, such as request a new form definition, exit program, perform a database update with the new data entered on the terminal, etc.

The user must not write an application program that makes use of function keys unless all terminals that are to run the program are equipped with function keys.

▶ ##FORCEREAD

The ##FORCEREAD directive forces FORMS to wait for and to process user input from the terminal, thus providing a facility to override the normal input protocol when processing a form definition with multiple substreams. Normally, terminal input occurs when the application program (1) executes the first read statement after the form is invoked, (2) issues a read statement following a write statement, or (3) attempts to read a substream that has already been read. The FORCEREAD directive causes terminal input on the next read operation, whether or not the next substream to be processed has already been read.

##FORCEREAD Example

```
          WRITE (1,200)
    200   FORMAT ('##FORCEREAD')
          READ (1,210) IREC
    210   FORMAT (32A2)
```

▶ ##INVOKE formname

The ##INVOKE directive defines the form definition to be used. It is followed by the form (stream descriptor) name, formname.

When the ##INVOKE directive is issued, FORMS searches the catalog for the specified form definition. If found, it is read into memory and initialized. If not found, an error message is printed and return is made to PRIMOS command level. When a form is invoked on a device, all input and output requests for that device are trapped and handled by the run-time package. When the form definition is subsequently released, all I/O is handled by calls to IOCS subroutines. (Refer to the Subroutine Reference Guide.

If a previous form definition was invoked and not released before issuing the current ##INVOKE directive, an implied release of the previous form definition occurs.

##INVOKE Example

```
        WRITE (1,120)
120     FORMAT ('##INVOKE TAXD01')
```

▶ ##POSITION

The ##POSITION directive allows the specification of the field to which the cursor will be positioned on the next read operation. This command is only applicable to the next read operation; subsequent read operations position the cursor to the first NOPROTECTed character position on the terminal, unless subsequent ##POSITION commands are issued.

##POSITION Example

```
C--- POSITION TO FIELD SPECIFIED BY CONTENTS OF 'FLDNAM'.
C
        WRITE (1,250) FLDNAM
250     FORMAT ('##POSITION ', 4A2)
```

▶ ##PRINT formname [LOCAL]

The ##PRINT directive allows the user to print the current form and user-entered data from the terminal to either the spooled line printer or a local printer attached to the individual terminal. This permits the program to print the current transaction on a hard-copy device without defining a separate FORMAT descriptor for the line printer.

If the parameter LOCAL is included in the ##PRINT directive, the form is printed on whatever hardcopy device is attached to the terminal. If the LOCAL parameter is not given, the copy is spooled for printing on the line printer. If a special paper form is needed by the line printer, it must be mounted by the operator.

##PRINT Examples

```
        WRITE (1,200)              /*WRITE TRANS. TO SYS PRINTER
200     FORMAT ('##PRINT')
```

##RELEASE

The ##RELEASE directive specifies that the current form definition is no longer to be used. All I/O is processed via calls to IOCS until the next ##INVOKE directive.

##RELEASE Example

```
        WRITE (1,900)
900     FORMAT ('##RELEASE')
```

##SUBSTREAM streamname

The ##SUBSTREAM statement defines the substream to be processed on the next READ or WRITE directive in the host language. The SUBSTREAM name streamname must follow the directive and be separated from it by at least one space. If the named SUBSTREAM does not exist in the STREAM descriptor, an error message is generated and the program aborts.

##SUBSTREAM Example

```
        WRITE (1,200)
200     FORM ('##SUBSTREAM NAMADDR')
```

##VALIDATE

The ##VALIDATE directive causes the run-time package to return the validation status of all input data on encountering the next READ statement(s).

The status is returned in the form of a two-digit number for each input/empty-conditional (see Section 5) or direct field (see Section 5) that is not declared as output-only. It is not returned for input-literal fields. The two-digit number returned represents one of the following conditions:

Number Range	Condition
<0	The data failed all validation tests. (Value usually -1, in this case.
00	No validation specified for this field.
>0	This is the number of the first validation mask that the data passed. Validation masks are numbered in the order in which they appear in the FIELD definition.

The validation status is returned in the same manner that data is returned on a READ statement. If there are multiple SUBSTREAM definitions, the user must do multiple READs to input the validation status for all fields. The ##VALIDATE directive causes the next READ statement to input the validation status of the first SUBSTREAM in the STREAM descriptor unless a ##SUBSTREAM directive is issued before ##VALIDATE. The ##VALIDATE function is disabled and normal data input resumed when either the end of the STREAM descriptor is encountered or a ##SUBSTREAM or ##FORCEREAD directive is issued.

VALIDATE Example

```
C--- INPUT FIELD VALIDATION.  FIRST SUBSTREAM CONTAINS 5 INPUT
C    FIELDS, SECOND CONTAINS 4.
C
       WRITE (1,300) /* POS TO FIRST SUBSTREAM
300    FORMAT ('##SUBSTREAM ONE'/'##VALIDATE')
       READ (1,310) (IVAL(I),I=1,9)  /* READ VALIDATION
310    FORMAT(5I2/4I2)
```

ATTRIBUTE MODIFICATION DIRECTIVES

The application program may dynamically change the attributes of a field by issuing one of the attribute directives. From one to twenty stream descriptor field names may be placed as arguments, each separated by at least one space. The modification occurs at the next WRITE or READ in which data (as opposed to FORMS directives) is transferred to or from the device.

The following table describes each of the eight attribute-modification directives and three synonyms.

Statement/Synonym	Description
##PROTECT	write-protects field
##NOPROTECT/ENABLE	write-enables field
##RVIDEO	field displayed on reverse video
##NVIDEO	field displayed in normal video
##BLINK	blinks field when displayed
##NOBLINK	field is displayed when form is output
##DISPLAY/FREE	field is displayed when form is output
##NODISPLAY/HOLD	field is not displayed when form is output

Example:

```
      WRITE (1,300)
300   FORMAT ('##PROTECT NAME, IDNUMBER, ADDRESS'/
     +      '##NOPROTECT REMARK1 REMARK2 REMARK3')
```

PROGRAMMING CONSIDERATIONS

Run-time File Handling

FORMS usually requires only one file unit for all file I/O. This unit number is assigned dynamically and is the first available file unit not already open.

The only exception is for the system printer device driver. To copy files into the spool queue, two file units are required. Units 15 and 16 are used by PR$IO. To allocate two other units, modify variables FUNITC (=5), PRINFO(1) (=15), and PRINFO(2) (=16) in the file FORMS>IOS>PRO$IO. All are declared in DATA directives.

Run-time Error Handling

Run-time error diagnostics generated by FORMS are self-explanatory.

Error message text is stored in the file FORMS*>RUN.ER. Each line in the file is preceded by a numeric key, between 1 and 9999. If a diagnostic requires more than one line (as do most), each following line contains the same numeric key as the first. The end of the diagnostic occurs when a line with a different numeric key is encountered.

The procedure that calls the FORMS error handler may supply from 0 to 3 arguments. An argument is inserted into the error diagnostic when a percent sign (%) followed by a value (1-3) is encountered in the text string.

The calling sequence for FORMS run-time error handler is:

```
CALL FM$ERR (KEY, FSCODE, TEXT1, LEN1, TEXT2, LEN2, TEXT3, LEN3)
```

Parameter	Memory
KEY	Numeric key of the error diagnostic to be printed. If this diagnostic is not included in the file, an error message is printed containing the error number. All errors generated by FORMS have corresponding text messages in RUN.ER.
FSCODE	The file system code associated with this error condition. If this error is not a result of a file system error, this value should be zero.
TEXTn	Text argument n. If not referenced by the error diagnostic, this need not be supplied.
LENn	Length in characters of the corresponding text argument. If this argument is not used, it may be omitted.

Users who write their own device drivers may make use of this error handling facility. Numeric error codes (keys) 1-999 are reserved. Users may allocate any error code above, and including, 1000.

Loading the Shared Library

It is a system administration decision whether or not to support the shared libraries. If shared FORMS is supported, shared COBOL, MIDAS (KI/DA on master disk), and FORTRAN must be supported as well. If shared libraries are in use, the FORMS shared library file will be named VFORMS.

Configurable I/O List

The FORMS run-time package contains a fixed length buffer, called the I/O list that holds the current form definition. The default I/O list size is 2500 words (decimal). If the user runs a program which invokes a form that exceeds this capacity, FORMS prints the error message:

```
REQUIRED= nnnn, AVAILABLE= 2500.
I/O LIST OVERFLOW.
```

The user may allocate a larger I/O list in his (FORTRAN) program by inserting the following three statements:

```
PARAMETER IOLSIZ=desired_size
COMMON /IOBCM$/ IBUF(3), IOL(IOLSIZ)
DATA IBUF /IOLSIZ, 0, 0/
```

All items are 16 bit integers.

The COBOL user may enlarge the I/O line size used in his COBOL program by writing a short FORTRAN subprogram (statements start in column 7):

```
SUBROUTINE name
PARAMETER IOLSIZ=desired_size
COMMON /IOBCM$/IBUF(3),IOL(IOLSIZ)
DATA IBUF/IOLSIZ,0,0/
END
```

Compile this subprogram with the FORTRAN compiler. When loading the main COBOL program, load this FORTRAN subprogram's binary module (B_name) containing the redefinition of the I/O list prior to loading the FORMS library.

The user may also modify the default buffer pool size by changing the 'IOLSIZ' declaration in 'FORMS>RUN>IOLDEF' and re-compiling the run-time system, using the appropriate tools provided on the master disk. Refer to the System Administrator's Guide. (This should be done under the direction of a system administrator or senior systems analyst.)

This feature is not available when using the 64V mode shared FORMS library; however, the default size for the shared library is 7000 words.

SECTION 5

FORMS DEFINITION LANGUAGE, REFERENCE INFORMATION

This section defines the data definition language (FDL) that is used to describe to the FORMS system the data formats discussed in Sections 1 through 3.

SYNTAX OF FORMS DEFINITION LANGUAGE

Statements

FDL supports a free-FORMAT statement line.

All descriptor, substream, and field names start in the first character position of the line and are followed by at least one space. Data STREAM and device FORMAT descriptor statements may start anywhere after column 1 and occupy columns 2 through 72. Columns 73 through 80 are ignored. Items in the FDL input line must be separated by either a space or a comma unless otherwise noted. Lower-case characters are mapped to upper-case, except characters in a literal string (enclosed within single quotes).

If an input record contains too many characters to fit on one line, source text may be continued by placing a semicolon (;) at the end of the line. Input items (words, text strings, etc.) may not be split across two lines. There is no limit to the number of continuation lines in a source record. There is, however, a 240 character limit per statement.

Comments

If the first character of a line is an asterisk, the line is treated as a comment, listed in the output file and ignored. If the first character is a single quote (') the line is treated as a comment, but this line causes an eject page in the listing and becomes the new page header.

In addition to full-line comments (lines beginning with an asterisk or single quote), in-line comments are supported. In-line comments are preceded by a fore-slash and asterisk (/*) and followed by an asterisk and a fore-slash (*/). If the in-line comment is the last item on the line, the terminating characters (*/) may be omitted. In-line comments may not occur within an item (e.g., in the middle of a name or text string).

Examples of Comments are:

```
* THIS IS A COMMENT LINE

' THIS WILL CAUSE A PAGE-EJECT AND WILL BECOME THE NEW HEADER

LABEL  FIELD ABC, LENGTH 6    /* THIS IS AN IN-LINE COMMENT

LABEL  FIELD ABC, /*THIS TOO IS AN IN-LINE COMMENT*/ LENGTH 6

NAME    FIELD 'FOUR SCORE AND SEVEN YEARS AGO... ' ;
        POSITION (10,10) PROTECT /* CONTINUATION LINE
```

Naming Conventions

The rules for naming FORMS descriptors, fields, and substreams are:

- name length: 1-8 characters
- first character must be alphabetic
- permitted characters: A-Z, 0-9

Examples of form descriptor names are:

Example	Comment
GAZORKLEFORM	name too long
SHIPFORM	valid
5FORM	bad first character (5)
FORM5	valid
OWED$	illegal character ($)
AMTOWED	valid

Descriptor Structure

Figure 5-1 represents various form definition structures for both the STREAM descriptor and FORMAT descriptor (refer to Section 3 for an overview of these descriptors).

Each statement illustrated in Figure 5-1 is detailed in the paragraphs that follow.

FORM DEFINITION DELIMITER STATEMENTS

The FDL statements described in this section are used to specify the beginning and end of a form definition or a section of a form definition. These statements do not describe data formats but rather are used to identify STREAM and FORMAT descriptors, SUBSTREAM descriptions and DEVICE descriptions within a FORMAT descriptor.

Stream Descriptor

```
STREAM statement                      STREAM statement
      .                               SUBSTREAM statement
      .                                     .
      .                                     .
FIELD definitions        -or-               .
      .                               FIELD definitions
      .                                     .
      .                                     .
END STREAM statement                        .
                                      END SUBSTREAM statement
                                      SUBSTREAM statement
                                            .
                                            .
                                            .
                                      FIELD definitions
                                            .
                                            .
                                            .
END SUBSTREAM statement
END STREAM statement
```

FORMAT Descriptor

```
FORMAT statement
 DEVICE statement 1
        .
        .
        .
  FIELD definitions
        .
        .
        .
END DEVICE statement
 DEVICE statement 2
        .
        .
        .
  FIELD definitions
        .
        .
        .
END DEVICE statement
END FORMAT statement
```

Figure 5-1. Form Definition Statements.

STREAM Definition Statements

▶ name STREAM

The STREAM statement defines the beginning of a STREAM descriptor. The name field begins in column 1 and must contain a unique stream descriptor name (i.e., one that does not conflict with any other stream descriptor defined within the system). For example:

```
SHIPFORM   STREAM
```

▶ END STREAM

The END STREAM statement defines the end of a STREAM descriptor. For example:

```
END STREAM
```

▶ name SUBSTREAM

The SUBSTREAM statement defines the beginning of a substream description. The application program transfers data to and from the substream specified by name. (Refer to the introduction to substreams in Sections 3 and 4, and the example using substreams in Section 7, For example:

```
USERDATA  SUBSTREAM
```

▶ END SUBSTREAM

The END SUBSTREAM statement terminates a substream description. Each SUBSTREAM statement must have an associated END SUBSTREAM statement. For example:

```
END SUBSTREAM
```

FORMAT Definition Statements

▶ name FORMAT

The FORMAT statement defines the beginning of a FORMAT descriptor. The contents of the name field defines the name of the format descriptor. The name must be equivalent to that of the stream descriptor with which this FORMAT descriptor will be used. For example:

```
USERDATA   FORMAT
```

▶ END FORMAT

The END FORMAT statement terminates the FORMAT descriptor and must be the last statement in the FORMAT description. For example:

```
END FORMAT
```

DEVICE Definition Statements

▶ DEVICE dev

The DEVICE statement specifies the name of the device that is defined by the field definitions in the following FIELD statements. dev is the specified device name. This statement is used in the FORMAT descriptor immediately following a FORMAT or END DEVICE statement. For example:

```
ADMN377 FORMAT
    DEVICE VISTAR3
```

▶ END DEVICE

The END DEVICE statement defines the end of a device description within a FORMAT descriptor. For example:

```
END DEVICE
```

FIELD STATEMENTS WITHIN A STREAM DESCRIPTOR

The fields defined within the STREAM descriptor by the FIELD statement identify: first, the location of each data item within the input or output record; second, its length; and third, optional justification and validation information.

Stream fields may be defined to be INPUT only, OUTPUT only, or INPUT-OUTPUT, which is the default value. INPUT fields are processed on input operations only, they are ignored on output. The reverse is true for OUTPUT fields. INPUT-OUTPUT fields are processed on both input and output operations. Using INPUT and OUTPUT fields, the programmer may describe separate input and output record formats in a single STREAM descriptor.

FIELD Types

There are six types of STREAM descriptor fields. Each type either describes an item within the user's data record or describes a literal string to be mapped to a field defined in the FORMAT descriptor. The field types are described in the following paragraphs.

Direct: A direct field maps the data item in the input or output record to the named FORMAT descriptor field. Its format is as follows:

```
FIELD fieldname
```

Input Literal: An input literal field returns a literal string to the data record on an input operation. It is ignored on output operations. Its format is as follows:

```
FIELD 'literal text string'
```

Output Literal: An output literal field defines a literal text string to be mapped into a FORMAT field on output operations. It is ignored on input operations. No data is transferred to or from the input/output record. Its format is as follows:

```
FIELD (fieldname,'literal text string'),OUTPUT
```

Input Empty Conditional: An input empty conditional (IEC) field functions the same as an input or an input-output direct field with one exception - if the data field displayed on the device contains spaces, the supplied literal string is returned instead of blanks. IEC fields may not be output only. Also, IEC fields require an input-output specification (the value of iospec may be INPUT or INPUT-OUTPUT). Its format is as follows:

```
FIELD (fieldname,'literal text string'), iospec
```

Filler: Fields defined as fillers perform no data transfer between the application program and the device. They only define a gap in the input or output record. On input and/or output operations, the number of characters designated by the LENGTH parameter in the filler field definition are skipped. A filler field's format is as follows:

```
FIELD FILLER, LENGTH n
```

System Information: A system information field (SIF) acts like an output literal field. It is processed only on output operations and it maps data into a selected FORMAT field. The mapped data, however, is not a literal text string but a system related piece of information like: current time, data, user name and number, or form name. The format of a SIF field is as follows:

```
FIELD (fieldname,SIFname)
```

SIF names, contents, format, and length are described below:

DATE1:	date, YY/MM/DD	{8 characters}
DATE2:	date, DD-MMM-YY	{9 " }
DATE3:	date, MM/DD/YY	{8 " }
DATE4:	date, DD.MM.YY	{8 " }
TIME1:	time, HH:MM	{5 " }
TIME2:	time, HH:MM xM	{8 " }
USERNAME:	user login name, XXXXXX	{6 " }
USERNUM:	user number, NN	{2 " }
FORMNAME:	form name, XXXXXXXX	{8 " }

Each field with a direct, an output literal, an empty conditional, or a system information field type is identified by a one-to-eight-character name that must be unique within this STREAM definition. This name may be supplied in the left margin of the field definition statement. If not explicitly defined, the field name is assumed to be the name of the format field to which the STREAM field is mapped. To modify any attributes of this field, the field name is given as a parameter to a FORMS run-time directive (refer to Section 4).

FIELD Parameters (for STREAM)

The parameters discussed in the following paragraphs are position independent. They may appear in the field definition after the mapping or literal specification.

LENGTH Parameter: The LENGTH parameter defines the number of characters contained in the field. The keyword LENGTH must be followed by a positive non-zero integer.

LENGTH Usage: The LENGTH parameter when used with the various field types has different effects with respect to its usage being required, optional, or assigned default values. These are summarized in the following list.

field type	remarks
direct	required
input-literal	optional - if omitted, defaults to text string length; if supplied, text string is padded/truncated as required to meet given length
output-literal	same as input-literal
empty-conditional	same as input-literal
filler	required
system-info	ignored

JUSTIFY Parameter: The JUSTIFY parameter defines the justification to occur when any data is logically moved to this field. It must be followed by one of the following key words:

NONE	specifies no justification
LEFT	the field is left justified, right padded
RIGHT	the field is right justified, left padded
CENTER	the field is centered

Note

'JUSTIFY NONE' has the same effect as not specifying the JUSTIFY parameter.

If justification is specified on both the STREAM and FORMAT descriptor fields, the data is justified according to the STREAM descriptor field specification on input and according to the FORMAT descriptor field on output.

JUSTIFY Usage: The JUSTIFY parameter when used with the various field types has different effects with respect to its usage being required, optional, or assigned default values. These are summarized in the following list.

field type	remarks
direct	optional
input-literal	optional
output-literal	optional
empty-conditional	optional
filler	ignored
system-info	ignored

SPACE-FILL and ZERO-FILL Parameters: SPACE-FILL and ZERO-FILL are mutually exclusive parameters that define the fill character to be used when performing left or right justification. For each character position the data is shifted, either a space or zero is supplied on the end from which the shift is taking place. If the user enters the field with the data already right justified and right justification with zero-fill is specified in the form definition, left-most spaces (if any) will not be replaced with zeroes. If neither parameter is specified, SPACE-FILL is assumed.

SPACE-FILL and ZERO-FILL Usage:

See JUSTIFY Usage.

INPUT, OUTPUT, and INPUT-OUTPUT Parameters: The INPUT, OUTPUT, and INPUT-OUTPUT parameters, which are mutually exclusive, define the direction of data transfer in which the specified field is to be processed.

INPUT/OUTPUT/INPUT-OUTPUT Usage: The INPUT, OUTPUT, and INPUT-OUTPUT parameters, when used with the various field types, have different effects with respect to their usage being required, optional, or being assigned default values. These are summarized in the following list.

field type	remarks
direct	optional; default is INPUT-OUTPUT
input-literal	default to INPUT, if specified, must be INPUT
output-literal	must be specified as OUTPUT
empty-conditional	must be specified as INPUT or INPUT-OUTPUT
filler	optional; default in INPUT-OUTPUT
system-info	ignored

VALIDATE Parameter: The VALIDATE parameter defines the validation to take place on the field data when read from the device. The keyword VALIDATE is followed by one or more validation masks enclosed in single quotes and optionally separated by the word 'OR'.

When a field with a validation specification is transferred to the input record at run-time, the data is checked against the validation mask(s) supplied by the user. If all of the validation tests are passed, the next field is transferred to the input record. If the data fails all tests, FORMS performs one of two actions specified by the FIX/NOFIX parameters.

A validation mask consists of a string of characters, each defining a certain criterion for the corresponding character in the field. If the length of the validation mask is less than that of the data field, the last character of the validation mask is logically repeated until the data field is exhausted.

The validation mask characters and their meanings are:

Mask Character	Validation Criteria
9	numeric (Ø-9)
A	alphabetic (A-Z, a-z)
X	alphanumeric (Ø-9, A-Z, a-z)
.	period
/	fore-slash
B	space (blank)
$	dollar sign
-	dash
	any character
N	numeric character (Ø-9, +, -, or blank)
F	floating numeric (Ø-9, +, -, ., blank)
U	unsigned integer (Ø-9, blank)
P	personal name (A-Z, a-z, ., ', or blank)
Z	alphabetic character or space

VALIDATE Usage: The VALIDATE parameter when used with the various field types has different effects with respect to its usage being required, optional, or assigned default values. These are summarized in the following list.

field type	remarks
direct	optional
input-literal	ignored
output-literal	ignored
empty-conditional	optional
filler	ignored
system-info	ignored

For example: VALIDATE '99.99' or '99AA'

FIX, NOFIX Parameters: When a field with one or more validation masks fails to meet any validation criterion, the programmer has a choice of forcing the FORMS user to correct the data before FORMS returns it to the application program.

If FIX is specified, the data must pass one or more of the supplied validation tests before it is returned to the application program. If the data fails all validation tests, FORMS prints an error message in the lower right corner of the terminal display and positions the cursor to the first character position of the field in error. The user at the terminal may then correct the error and re-transmit the information.

If the NOFIX parameter is specified, the data is returned to the program whether or not it passes any of the validation tests. When the input record is complete, FORMS returns to the error return location instead of taking the standard return. An ERR= clause must be present in a FORTRAN read statement if any fields in the form definition contain a paragraph to do "error" processing. A validation error may be identified by either a FORTRAN or COBOL program by inspecting the two-character error code in the error vector by calling the GETERR system subroutine. (Refer to Subroutine Reference Guide). FORMS sets this code to VA for validation errors.

In most cases, it is convenient to require the data be in proper format when it reaches the application program (i.e., using the FIX parameter), eliminating the task of inspecting multiple fields on a character-by-character basis.

If FIX or NOFIX is not specified, FIX is assumed.

FIX/NOFIX Usage: The FIX or NOFIX parameter when used with the various field types has different effects with respect to its usage being required, optional, or assigned default values. These are summarized in the following list.

field type	remarks
direct	optional
input-literal	ignored
output-literal	ignored
empty-conditional	optional
filler	ignored
system-info	ignored

START Parameter: The START parameter allows the user to specify the character position occupied by the first character of the data field within the input or output record. The the START parameter function is equivalent to that of the 'T' FORMAT descriptor in a FORTRAN FORMAT statement. START allows overlapping of input/output fields, a function not available with the FILLER specification.

The word START must be followed by an integer number that represents the absolute character position (within the user's data record) of the first character of the field.

CAUTION

> If START is specified in an input-only field, the character pointer gets reset for the input record but not for the output record. The inverse is true for output-only fields. This is reflected in the input and output STREAM descriptor formats generated by FDL, if the -IOFLIST option is specified.

START Usage: The START parameter when used with the various field types has different effects with respect to its usage being required, optional, or assigned default values. These are summarized in the following list.

field type	remarks
direct	optional
input-literal	optional
output-literal	optional
empty-conditional	optional
filler	optional
system-info	ignored

FIELD DEFINITION EXAMPLES: STREAM DESCRIPTORS

The following FDL coding example shows how the types of STREAM discussed in this section are defined. The lines of code also show usage of some field parameters discussed. Before each field definition, a comment line has been inserted that identifies the type of field being defined.

The following example code shows and example of definition of each type of field discussed and use of some of the field parameters that are significant when used with the particular field types.

```
*   DIRECT FIELD TYPE.
*
           FIELD IDNUM, LENGTH 5
*
*   LITERAL INPUT FIELD, RETURN STARTING IN COLUMN 30
           FIELD 'LITERAL INPUT STRING', START 30
*
*   OUTPUT LITERAL FIELD
           FIELD (HEADER,'HEADER TEXT'), OUTPUT
*
*   INPUT EMPTY CONDITIONAL FIELD TYPE
           FIELD (EMPLNAME,'NO EMPLOYEE NAME SPECIFIED'), ;
                INPUT-OUTPUT
*   NOTE USE OF SEMICOLON FOR FDL CONTINUATION LINE
*
*   FILLER FIELD
           FIELD FILLER, LENGTH 12
*
*   SYSTEM INFORMATION FIELD
           FIELD (OUTDATE,,DATE3)
*
*   INPUT-ONLY FIELD, JUSTIFY AND VALIDATE
           FIELD AGE, LENGTH 3, JUSTIFY RIGHT,, INPUT, ;
                VALIDATE '999' OR 'B'
```

FIELD STATEMENTS WITHIN A FORMAT DESCRIPTOR

Fields defined within the FORMAT descriptor describe the appearance of data on the page-oriented device. This definition includes (1) field coordinates, (2) length, (3) justification, and (4) any display attributes to be associated with the data (e.g., write enable/protect, blink, reverse video, etc.).

FIELD Types

There are two types of FORMAT descriptor field statements: mapped and literal. These are described in the following paragraphs.

Mapped: A mapped field is actually mapped to by a field defined in the STREAM descriptor. All mapped fields contain a 1-8 character name starting in the left margin in the field definition statement. The format of a mapped field is as follows:

```
name FIELD field-name
```

Any mapped field defined in the FORMAT descriptor and not mapped to by a STREAM field is ignored. Any STREAM descriptor field that maps to a nonexistent FORMAT descriptor field is also ignored.

Literal: A literal field contains a text string specified in the FORMAT descriptor field definition. Literal fields are used to supply tags (titles) for information displayed on the device and usually identify mapped fields. The format of a literal field is as follows:

```
FIELD 'literal text string'
```

The literal data is specified immediately following the FIELD statement and must be enclosed within single quotes. The name field (left margin) must be blank (no mapping is done from a stream descriptor field.)

FIELD Parameters (for FORMAT)

The following parameters may follow the FIELD statement in a mapped field and the literal specification in a literal field. They are all non-positional (i.e., they may occur anywhere in the field definition). All parameters apply to both the mapped and literal device descriptor field types. All parameters are optional unless otherwise noted.

LENGTH Parameter: The LENGTH parameter defines the length of the field as it is to appear on the device. It must be followed by a positive non-zero integer that represents the field length in characters. This parameter is required on mapped fields and is optional on literal fields. If omitted, the field length is assumed to be the length of the literal string.

The length of a field in the STREAM descriptor may differ from the length of a field in the FORMAT descriptor. The STREAM field defines the length in the input/output record of the application program and the device format field length defines the length of the field on the input/output device. If they differ, the data is truncated or padded as required.

POSITION Parameter: The POSITION parameter defines the position (column and line) of the first character in the field. The keyword POSITION is followed by the column and line (x,y) address, enclosed within parentheses and separated by a comma. This parameter is mandatory on both mapped and literal fields.

JUSTIFY Parameter: The JUSTIFY parameter defines the justification to take place when data is logically moved to (through) this field. Refer to the description of the STREAM descriptor field JUSTIFY parameter. This parameter is optional on both mapped and literal fields, and its default value is JUSTIFY NONE if JUSTIFY is not specified.

Display Attribute Parameters

The following eight parameters are used to describe the display characteristics of the field data when it is written to the specified device. If a device does not support a feature, such as reverse video or blink, the attribute is ignored.

NOPROTECT (or ENABLE) Parameter: The NOPROTECT parameter, which is mutually exclusive with PROTECT, declares the associated field to be write-enabled upon display at the user terminal. When displayed on the line printer, the field is underlined (if underlining is available).

PROTECT Parameter: The PROTECT parameter declares that the field is to be displayed write-protected when written to the user terminal. When printed at the line printer, it is not underlined (displayed normally). If neither PROTECT nor NOPROTECT is specified, PROTECT is assumed.

BLINK Parameter: The BLINK parameter defines the field to be blinked when displayed on the terminal. It has no effect in a device descriptor for the printer.

NOBLINK Parameter: The NOBLINK parameter defines the field as not blinked when displayed at the user terminal. If both BLINK and NOBLINK are omitted, the default value is NOBLINK.

REVERSE VIDEO Parameter: The REVERSE VIDEO parameter causes the field to be displayed in reverse video at the user terminal. It has no effect when output is printed at the line printer.

NORMAL VIDEO Parameter: The NORMAL VIDEO parameter declares the field to be displayed in normal video at the terminal. If both the REVERSE VIDEO and NORMAL VIDEO parameters are omitted, the default value is NORMAL VIDEO.

NODISPLAY (or HOLD) Parameter: The NODISPLAY parameter causes the field not to be displayed when the form is output. It is valid on all terminal and line printer device types.

DISPLAY (or FREE) Parameter: The DISPLAY attribute causes the field to be displayed when the form is output to either the terminal or the line printer. If both the DISPLAY and NODISPLAY parameters are omitted, the default value is DISPLAY.

FIELD DEFINITION EXAMPLES: FORMAT DESCRIPTOR

The following FDL coding example shows how the types of FORMAT fields discussed in this section are defined. The lines of code also show usage of some field parameters discussed. Before each field definition, a comment line has been inserted that identifies the type of field being defined.

```
*   MAPPED FIELD, NOT WRITE-PROTECTED
*
INVNUM      FIELD POSITION (70,2), LENGTH 6, NO PROTECT

*   LITERAL FIELD
*
            FIELD 'Literal String Test', POSITION (1,4) ;
```

PROGRAMMING AIDS

The following paragraphs describe FDL statements to assist the programmer designing a form. They include a macro capability (the DEFINE statement) and iterative field generation (the REPEAT statement).

MACRO DEFINITION

Currently, a macro consists simply of one text item replacing another item or text string (i.e., a synonym).

name DEFINE item

The DEFINE statement allows the definition of a macro.

A DEFINE statement must be preceded by the name of the macro, starting in Column 1. The statement name must be followed by one or more spaces and then by the macro text.

Whenever the macro name is encountered as a single item within an input line (not in a literal text string), the macro name is replaced by the given definition. All macros must be defined before they are used.

Macro definitions are not retained between form definitions. They are erased after each END STREAM and END FORMAT statement.

Examples

```
FLD       DEFINE   FIELD
LEN       DEFINE   LENGTH
POS       DEFINE   POSITION
D1X       DEFINE   5
D1Y       DEFINE   10
*
*
*   FIELD DEFINITION USING ABOVE MACRO DEFINITIONS
DATA1     FLD, POS (D1X,D1Y), LEN 10
*
*   NOTE THAT THIS HAS THE SAME FUNCTION AS:
DATA1     FIELD, POSITION (5,10), LENGTH 10
```

ITERATIVE FIELD GENERATION

Iterative field generation allows generation of multiple blocks of field statements with only one block definition. Fields to be generated in this manner must be enclosed within REPEAT and END REPEAT statements.

Iterative field generation is permitted in both STREAM descriptor and FORMAT descriptor definitions. A two-digit iteration number is appended to any field name found in either the left margin or immediately following a STREAM field statement. If the field name is seven or eight characters, it is truncated to six characters to permit the iteration number to be appended. The same is true for FORMAT (mapped) field names encountered in direct, output-literal, and input/empty-conditional STREAM descriptor fields.

▶ REPEAT n

The REPEAT statement defines the beginning of an iterative field generation (REPEAT) block. It must be followed by an integer (n) greater than zero that represents the number of iterations to make through the field definitions that follow in the FDL source text. The iteration counter is initially set to one and is incremented by one at each pass through the REPEAT block. When the counter exceeds the specified repeat count, the statement immediately following the associated END REPEAT statement is processed.

Only FIELD statements are permitted within a REPEAT block.

▶ END REPEAT

The END REPEAT statement terminates a REPEAT block. For each REPEAT statement there must be a corresponding END REPEAT statement. Repeat blocks may not be nested.

RELATIVE POSITION PARAMETER

A second form of the POSITION parameter is available to fields defined within a repeat block. This permits the field coordinates to be relative to the current iteration number instead of absolute line and column.

Relative positioning is specified by placing a plus or minus sign immediately preceding the line and/or column definition in the POSITION parameter. The absolute line or column number is computed by adding or subtracting the current iteration number to or from the specified offset.

An example of both iterative field generation and relative positioning is shown in the following linew of FDL code:

```
*   THIS BLOCK WILL BE REPEATED 3 TIMES
*
    REPEAT 3
LASTNM    FIELD  LENGTH 20, POSITION (10,+7)
FRSTNM    FIELD  LENGTH 10, POSITION (35,+7)
MIDDIN    FIELD  LENGTH 1,  POSITION (50,+7)
    END REPEAT
```

The previously shown FDL code that used the relative position feature and a repeat block accomplishes the same function as the following FDL code that does not take advantage of these features:

```
LASTNM01  FIELD  LENGTH 20, POSITION (10,8)
FRSTNM01  FIELD  LENGTH 10, POSITION (35,8)
MIDDIN01  FIELD  LENGTH 1,  POSITION (50,8)
LASTNM02  FIELD  LENGTH 20, POSITION (10,9)
FRSTNM02  FIELD  LENGTH 10, POSITION (35,9)
MIDDIN02  FIELD  LENGTH 1,  POSITION (50,9)
LASTNM03  FIELD  LENGTH 20, POSITION (10,10)
FRSTNM03  FIELD  LENGTH 10, POSITION (35,10)
MIDDIN03  FIELD  LENGTH 1,  POSITION (50,10)
```

LISTING CONTROL STATEMENTS

▶ NOLIST

The NOLIST statement disables the listing of all FDL statements, macro and repeat block expansions, except for those containing errors. It is overridden only by the -EXPLIST command line option.

▶ EJECT

The EJECT statement causes the listing to eject to the top of a new page when the listing file is output (spooled) to the line printer. The old page header is retained. For a new page header, refer to the section entitled 'General Syntax'. The EJECT statement has no effect if the listing is turned off (via the ERRORS ONLY option or NOLIST statement).

ALTERNATE INPUT FILE ($INSERT)

▶ $INSERT pathname

The contents of another FDL source file may be inserted into the primary input file at translation time. This is accomplished by placing the $INSERT directive in the left margin of the input line, followed by at least one space, and then the pathname of the file to be inserted. Input is then obtained from the inserted (alternate) disk file until the end of file (EOF) is encountered. When EOF is reached, FDL resumes processing the primary input file at the line following the $INSERT directive. No modification of the main input file is done. This temporarily "switches" the input flow from the primary to the alternate input file.

The $INSERT directive provides a convenient method of incorporating a common macro definition file into an FDL source file. For example:

```
$INSERT &lt;SOFTWR&gt; FORMS&gt; MACROS
```

FDL TRANSLATION, COMMAND FORMAT

▶ FDL pathname -options

FDL is invoked by entering the external command FDL. The command may be followed by an input file name and/or a list of translation options.

pathname specifies the input (source) text if it is to be obtained from file specified by pathname. This parameter may only appear immediately following the command (in the option -INPUT) name.

Option	Definition
-INPUT pathname	defines the source file, same as pathname, but may appear anywhere on the command line
-INPUT TTY	source text is to be obtained from the user terminal
-LISTING	listing file is to be generated
-LISTING lpath	listing is to be written to file specified by the pathname lpath
-LISTING NO	no listing file is to be produced
-LISTING TTY	listing is to be printed at user terminal
-LISTING SPOOL	listing file is to be routed directly to spool queue - the name of the spool file is printed on the user terminal prior to start of translation
-BINARY	binary file is to be generated
-BINARY bpath	binary file output file is to be generated with the name specified by bpath
-BINARY NO	no binary file is to be generated

If a -BINARY or -LISTING option is not followed by a ipath or opath, the binary file is written to either the file open on File Unit 3 or to a file called B_filename, if no file is open. Similarly, the listing file is written to either the file open on File Unit 2 or to a file called L_filename, if no file is open.

The following are the FDL-specific options (minimum abbreviations are underlined):

option	definition
-OBJLIST	list emitted object text
-MACLIST	generate expanded macro listing
-ERRLIST	generate errors-only listing
-EXPLIST	override NOLIST pseudo-op
-ERRTERM	list errors on user terminal
-IOFLIST	list I/O and device formats
-REPLIST	expand and list repeat blocks

Each FDL-specific option except -EXPLIST may be preceded by a NO to reverse the option's meaning. For example, -NOMACLIST specifies that an expanded macro listing is to be suppressed. An optional parameter may be abbreviated to the minimum number of characters required to distinguish it from other parameters. For example:

```
FDL FDEF15 -LISTING SPOOL -BINARY NO -OBJ -MAC
```

Default Option Values

Each installation may choose a set of default options for the FDL translator. Currently, the following options are standard:

```
-LISTING  -BINARY  -IOFLIST  -ERRTERM
```

All other options are disabled. FDL defaults are set by the A-register setting in the translator's memory-image file. The user may select his own default options by RESTORE'ing a copy of FDL and SAVE'ing it with the desired bits set in the A-register. The following table shows the A-register bit settings for FDL options and device codes:

options

bit	set for
1	-OBJLIST
2	-MACLIST
3	-ERRLIST
4	-EXPLIST
5	-ERRTERM
6	-IOFLIST
7	-REPLIST
8-10	input device
11-13	listing device
14-16	binary device

device codes

code	device
0 >	none
1 >	terminal
2 >	paper tape
3 >	card reader
4 >	printer
5 >	magtape
6 >	undefined
7 >	disk file

The default A-register setting is '6777.

RUN-TIME MESSAGES

After each stream or format descriptor is translated, FDL prints a message at the user terminal containing the number of errors encountered in the source text and the FDL revision number.

FDL TEMPORARY FILES

While translating a source file, the FDL translator may produce one or more of the following files:

Name	Format	Contents
ER##uu	ascii	error definitions (*)
RP##uu	ascii	current repeat block
IN##uu	ascii	input stream/substream definition
OU##uu	ascii/ binary	output stream/substream format device format map

All files are created and deleted by FDL. The only way that the user can examine them is to use CNTL-P to break out of the translator and/or perform a LISTF while another user is running FDL in the same UFD.

* The <u>uu</u> in the filename denotes the current user number - this permits multiple FDL translations simultaneously within the same directory.

FDL COMMAND LINE EXAMPLE

```
OK, FDL DS1
GO
0000 ERRORS (FDL, REV 16 - 16-FEB-79)
0000 ERRORS (FDL, REV 16 - 16-FEB-79)

OK,
```

SECTION 6

FORMS ADMINISTRATIVE PROCESSOR (FAP), REFERENCE INFORMATION

FUNCTION

The FORMS Administrative Processor (FAP) provides the commands to create and maintain the forms definition catalog, configure new terminals and new device drivers into the FORMS system, and obtain the system status.

COMMAND FORMAT

FAP is invoked by typing the command: FAP. FAP prints a header line followed by the current revision number.

FAP COMMANDS

The following paragraphs describe the commands supported by FAP. All command names may be abbreviated to three characters.

Available FAP commands are:

- ADD
- CREATE
- GENERATE
- JOURNAL
- LINK
- LIST
- PURGE
- QUIT
- REPLACE
- TCB

▶ ADD filename [LIST | LIST UPDATES]

The ADD command enables the user to add form definitions to the FORMS catalog. The name of the binary form definition file (filename), generated by the FDL translator, must immediately follow the keyword ADD. This filename usually starts with B_. One binary file may contain more than one form definition (e.g., if there was one stream descriptor and one format descriptor with definitions for three devices, the binary file contains four form definitions). FAP considers each DEVICE descriptor defined under a single FORMAT descriptor to be a separate form.

The ADD command adds only new modules to the FORMS catalog. Any attempt to replace a form already residing in the FORMS catalog with the ADD command causes the new form definition to be ignored and a warning message to be printed at the user terminal.

The input (binary) file name may be followed by the parameter LIST or LIST UPDATES. If this is specified, all form definitions added to the FORMS directory are listed by name on the terminal.

When the binary file has been processed, the number of modules added and ignored (due to duplicate entries) is printed.

If any translation errors were generated by FDL, the message WARNING! form-name CONTAINS ERRORS is printed or displayed at the terminal. A binary form definition with translation errors will probably generate undesirable results at run-time. The user must correct the source file and retranslate it with FDL.

ADD Examples

```
ADD B0FM03
01 DEFINITION ADDED.
ADD B-FM04 LIST

  DEDUCT   STR              V00   ADDED
  DEDUCT   FMT   VISTAR3    V00   ADDED
  DEDUCT   FMT   PRINTER    V00   ADDED

03 DEFINITIONS ADDED.
*
```

▶ CREATE [DIRECTORY]

The CREATE (or CREATE DIRECTORY) command allows creation of a skeleton FORMS catalog.

If the FORMS directory does not exist, FAP requests a disk volume-id on which the UFD named FORMS* is to be created. The user must then enter the volume-id (DSKRAT name) of the pack/partition that will contain the FORMS directory. FAP then asks for the MFD owner password on this volume. After this information has been entered, the FORMS UFD, catalog, and terminal configuration files are created. The CREATE command produces an error message if the FORMS catalog already exists. To create a fresh copy, the old file must first be deleted using the TREDEL command under FUTIL.

If the FORMS* UFD is created with FAP by the user, the following files must be copied to this directory before executing an application program that uses FORMS:

DCF.AS   RUN.ER   DCF.BN

These files may be found in the FORMS* UFD as released on the master disk.

CREATE Example

The following is an example of CREATE command dialogue.

```
OK, FAP
GO

FAP REV 16      11 - FEB - 79

* CREATE
UFD "FORMS*" DOES NOT EXIST
SHALL I CREATE IT? YES
ENTER OWNER PASSWORD (IT WON'T ECHO): ABCDEF
THIS MFD IS FULL, TRY AGAIN.

ENTER DISK VOLUME-ID: SOFTWR
ENTER OWNER PASSWORD (IT WON'T ECHO) XXXXXX

TCB CREATED.
DIRECTORY CREATED.

*
```

On any input request within the CREATE dialogue, the user may input CNTL-C to abort creation and return to the FAP command level.

▶ GENERATE

The GENERATE command is issued when the device control file has been modified; normally, when a new device driver has been added to the system or a device driver has been removed. The GENERATE command creates three $INSERT files and one binary file in FORMS* directory. The files generated are as follows:

- DEVEXT - external declaration statements for run-time device drivers
- DEVDAC - 64R mode driver dispatch table
- DEVIP - 64V mode driver dispatch table
- DCF.BN - binary representation of the new device control file (DCF.AS)

The GENERATE command must be issued and a new run-time I/O package must be assembled each time the device control file (DCF) is modified.

▶ JOURNAL [filename | START ON filename | STOP]

The JOURNAL command allows the logging of transactions with the FORMS catalog in an ASCII file that can be printed. All ADD, REPLACE, PURGE (described below), and TCB (described below) transactions are recorded in the JOURNAL file.

The JOURNAL command may be used to enable or disable the logging function. To disable it, the command JOURNAL or JOURNAL STOP may be issued. To enable it, the command line JOURNAL filename or JOURNAL START ON <filename> may be issued.

JOURNAL Example

```
* JOURNAL LOG000
* ADD BF01
* 08 DESCRIPTIONS ADDED.
* JOURNAL STOP
*
```

▶ LINK {ALL | formname}

The LINK command allows users to upgrade to current software revisions without having to rebuild each form definition. It may also be used to recover from various form definition file inconsistencies when a run-time error dictates such recovery is necessary.

There are two LINK command forms. LINK ALL specifies that all form definitions contained in the FORMS directory are to be re-linked. If the LINK command is followed by a formname, only the specified form definition is linked.

Linking is the process that combines the STREAM and FORMAT descriptors into one form definition. This form definition is then stored in a file in the directory named FORMS>LNK.FD, for faster access at execution time. It is the linked form definition and not the individual stream and format descriptor that is used when a form is invoked at execution time (run-time).

The linked form definitions are transparent to the user. They are automatically created or updated by FAP when a form definition is added to or replaced in the catalog. The corresponding link file is deleted when a format descriptor is purged.

▶ LIST [FILE filename] [ON TERMINAL | ON FILE filename 2]

The LIST command causes all or part of the FORMS catalog to be listed by name and type. This may be followed by a form name specifier to selectively list a part of the catalog. If the form name specifier is omitted, the entire catalog is listed. If the phrase ON FILE filename 2 is included, the catalog listing is written to the specified file. If the phrase ON TERMINAL is specified, or if the ON FILE specifier is omitted, the listing is written to the user terminal.

The information in the catalog listing includes:

. formname, type, and device (if any)
. version number
. owner (login) name
. creation, last access, last modified dates
  (file output only)

LIST Example

```
* LIST

FORMS DIRECTORY ON THURSDAY, FEBRUARY 2, 1978 AT 9:45 PM

 NAME      TYPE    DEVICE    VER    OWNER

HDRFØ1     STR               VØ2    JIMW
HDRFØ1     FMT    VISTAR3    VØØ    JIMW
HDRFØ2     STR               VØØ    DAVEW
HDRFØ2     FMT    VISTAR3    VØØ    DAVEW

Ø4 ENTRIES.
*
```

▶ PURGE (formname-specification) |[LIST UPDATES]

The PURGE command purges form definitions from the FORMS catalog. The PURGE command must be followed by a form name specification that designates which form definitions are to be purged. It may also be followed by the word LIST or LIST UPDATES, which causes all purged forms to be listed by name on the user terminal.

The formname-specification designates the form definitions to which the invocation of the PURGE command applies. Both PURGE and LIST commands use this option.

The formname-specification is enclosed in parentheses and has the following formats:

formname
formname.type
formname.type:device

type may be:

STR for stream descriptor, or
FMT for format descriptor

If only formname is specified, the PURGE command relates to all forms with the given name, any type and any device. If the second specification is used, the command relates to all forms of the given name and type. If the type is FMT, it relates to all device descriptors within the format definition. If the third type of specification is used, the command relates to the one definition that contains the same name, type, and device. This letter construction should only be used on format descriptors (there is no device definition for a stream descriptor!).

If any item in the form name specifier (formname, type, or device) is specified as an asterisk (*), or as the word ANY, no check will be made on that item when scanning the FORMS catalog.

Up to 20 formnames may be specified within the parentheses, separated by commas.

PURGE Examples

```
OK, FAP
GO

FAP REV 16   23-FEB-79


* PURGE (ORDER19)

   2 ENTRIES PURGED.

* PURGE (ORDER20.STR)

   1 ENTRY PURGED.

* PURGE (ORDER21.FMT:OWL1200)

   1 ENTRY PURGED.
```

▶ QUIT

The QUIT command causes FAP to exit and return to Primos command level. FAP may be re-entered by typing the START (S) command.

QUIT Example

```
* QUIT

OK,
```

▶ REPLACE filename

The REPLACE command functions the same as the ADD command, but causes any form definitions in the FORMS catalog that are redefined in the input (binary) file filename to be replaced with the new definition. Any form definitions in the binary file that are not defined in the catalog are added.

REPLACE Example

```
* REPLACE B-F019
  02 DEFINITIONS REPLACED.
* REPLACE B-F020
  01 DEFINITION ADDED   03 DEFINITIONS REPLACED.
```

```
        ( uu terminal     )
        ( * terminal      )
▶ TCB   ( LIST filename   )
        ( LIST            )
```

The TCB command modifies the terminal configuration file. This file contains a 64 by 4 word table that describes the terminal type for each FORMS user on the (local) computer system. It is used in conjunction with the device control file (DCF) at run-time to select the terminal device driver for a given FORMS user. Both TCB and DCF files are explained in detail in Appendix B.

To modify the terminal configuration file, the TCB command may optionally be followed by parameters that reflect the type of operation being performed. The parameter terminal specifies a type of terminal, such as OWL1200 or VISTAR3. The user number, uu when specified initiates an addition, a replacement, or a deletion from the Terminal Configuration file with regard to the user specified by uu.. When specified alone, uu causes the user with that number to be removed (deleted) from the block. If it is attempted to delete a nonexistant entry, a warning message and returns to FAP command level. If the value uu is not already in the terminal configuration file, and if

specified with a value for terminal, an addition is made to the file for user uu with terminal type terminal. However, if an entry for uu already exists in the terminal configuration file, and if a value for terminal is also specified, then a new entry replaces the existing entry for uu in the file. The TCB command with the parameter * in place of uu will cause the current values for uu to be placed in the terminal configuration file. TCB with the optional parameter LIST lists all the files in the FORMS catalog (directory). TCB LIST filename lists only those entries that pertain to filename. To add or change the terminal type, the 1- to 8- character terminal name must be input. If the specified user uu already had a TCB entry, the name of the old terminal type is printed on the terminal.

```
* TCB LIST

TERMINAL CONFIGURATION ON TUESDAY, JANUARY 9, 1979 AT 11:58 AM


USER    TERMINAL

 03     VISTAR3
 04     OWL1200
 05     OWL1200
 06     OWL1200
 07     OWL1200
 08     OWL1200
 09     OWL1200
 10     OWL1200
 11     OWL1200
 12     OWL1200
...

   32 ENTRIES.
...
```

FAP EXAMPLE

Having translated the example format and stream in Section 3 thus producing a binary file B_DATAS1 and a listing file L_DATAS1, the binary representation may be placed in the catalog by using the FAP command, as follows:

```
OK, FAP
GO

FAP REV 16   23-FEB-79

* ADD B_DATAS1
  1 DEFINITION ADDED
* LIST

FORMS DIRECTORY LISTING ON FRIDAY, FEBRUARY 23, 1979 AT 2:12 PM


  NAME      TYPE   DEVICE    VER   OWNER

PHONMENU    FMT    OWL1200   V08   DONL
PHONMENU    STR              V08   DONL
PHONFM      FMT    OWL1200   V08   DONL
PHONFM      STR              V08   DONL
PHONFM      FMT    VISTAR3   V00   DONL
PHONMENU    FMT    VISTAR3   V00   DONL
DATAS1      FMT    OWL1200   V00   JDOAKS
DATAS1      STR              V00   JDOAKS
FINCER01    STR              V18   STEVE1
FINCER01    FMT    OWL1200   V18   STEVE1


  11 ENTRIES.
* QUIT

OK,
```

SECTION 7

EXAMPLE FORTRAN PROGRAM

INTRODUCTION

This section describes the development of a FORMS application program from source coding to loading and execution. It also describes how to prepare the data that defines the form and how to place that data in the FORMS catalog. The complete source listing for the example is in Appendix E.

This example is based on a practical use of FORMS to keep track of customer orders, shipping information, and billable accounts. The program allows the user to input information at the terminal and store this information on a disk file. The application is typical of many parts distribution and billing operations. An example of the screen seen by the user is shown in Figure 7-1. Features that are essential to every FORMS application program as well as some special features unique to this application are discussed together with source code that illustrate these features. They are:

- Setting up data areas through use of standard FORTRAN statements (i.e., LOGICAL, INTEGER, EQUIVALANCE).
- Extending terminal I/O buffer size with calls to the Primos subroutine ATTDEV.
- Initializing control and output files (named ATS.C and ATS.D) with calls to the Primos subroutines SRCH$$ and PRWF$$.
- Initializing the descriptor for the form to be used by means of a formatted READ statement that transfers the FORMS directive ##INVOKE to the terminal device driver and causes the form to be initialized at the terminal i.e., displayed on the terminal screen).
- Reading the current control number from the control file, ATS.C, and updating the control file with the next control number.
- Identifying, to FORMS, the substream that is to be accessed next. ##SUBSTREAM is used in this case since the data input and output are contained in more than one logical record. If there were only one record, then the STREAM statement would be sufficient. When the header is written, the program sets up a substream for handling errror messages and two other substreams: one for general name and address information, and one for parts list information.

Figure 7-1. Screen Display Produced by Running Example Program.

- Accepting input data from the terminal using formatted READs and FORMS run-time directives.
- Positioning the control file to the next control number to be read, and clearing the terminal screen of the last block of data written to the output file.
- Exiting from the application program when done.

WRITING THE PROGRAM

Setting Up Data Areas

The following code illustrates how some of the data areas for the example program are defined.

```
       INTEGER NAMADR(75), VIA, HOW, REPL, INTC, BILL, SONUM(4),
      +        CHNNUM(4), CPO(4), ACOTHR(15), AIRSPR, INS(5),
      +        TYPE, CODE, NWIO, ATSNUM, B, I, J, MORE, FLD1(4,4),
      +        YESNOB(4), ACTBUF(4,3), FLD2(4,3)
```

NAMADR, for example, is intended to accomodate up to 75 words (or 150 ASCII characters) of name and address information.

Extending Size of Terminal I/O Buffers

The number of characters handled by a single transaction by this program (150) happens to be more than the default line size of the terminal read-write buffers (72). However, the size of the I/O terminal buffers may be changed by calls to the Primos subroutine ATTDEV. For example:

```
C --- EXTEND TERMINAL, FILE I/O BUFFERS.
C
      CALL ATTDEV(1,1,0,150)
      CALL ATTDEV(6,7,2,150)
```

The first call to ATTDEV sets up an input buffer, open for reading, from the terminal on File Unit 1. The second call to ATTDEV sets up an output buffer, open for writing on File Unit 6. In both cases, the size of the buffer is specified to be 150 characters.

Shared Library Initialization

In order to use the FORMS run-time directives with the shared libraries, the following call must be present in the application program prior to the first ##INVOKE directive:

```
      CALL FORM$I
```

This requirement is applicable to the 64V mode shared version of FORMS and is ignored for 64R mode and non-shared 64V mode.

Selecting a Library

It is a system administration decision whether or not to support the shared libraries. If shared FORMS is supported, shared COBOL, MIDAS, and FORTRAN must be supported as well. If shared libraries are in use, the FORMS shared library file is named VFORMS.

Initializing Control and Output Files

Calls to standard PRIMOS file system subroutines open, and read the control file (ATS.C) and open and position the output file (ATS.D) for writing. This is shown in the following code:

```
C --- OPEN FILES, INVOKE FORM ON TERMINAL.
C
      CALL SRCH$$(K$CLOS,0,0,1,0,CODE)
      CALL SRCH$$(K$CLOS,0,0,2,0,CODE)
C
C --- READ CONTROL FILE, ATS.C .
      CALL SRCH$$(K$RDWR,'ATS.C',5,1,TYPE,CODE)
C --- CHECK FOR FILE READ/WRITE ERRORS>
      IF (CODE.NE.0) CALL ERRPR$(K$NRTN,CODE,'ATS.C',5,0,0)
C --- OPEN AND POSITION DATR FILE, ATS.D .
      CALL SRCH$$(K$RDWR,'ATS.D',5,2,TYPE,CODE)
      IF (CODE.NE.0) CALL ERRPR$(K$NRTN,CODE,'ATS.D',5,0,0)
C
      CALL PRWF$$(K$POSN+K$PRER,2,LOC(0),0,10000000,NWIO,CODE)
```

The first two calls to the subroutine SRCH$$ close any files that might have been left open on File Units 1 and 2 before the execution of the application program started. The third call to SRCH$$ opens the file ATS.C for reading and writing on File Unit 1. The program makes provision for a standard error return. The fourth call to SRCH$$ likewise opens the output data file ATS.D for reading and writing on File Unit 2. Then, the call to PRWF$$ positions the output file to the next record to be written.

Initializing Form Descriptor

Every FORMS application program must identify the name of the form descriptor file installed in the FORMS catalog. This is accomplished by an ##INVOKE run-time directive embedded in a formatted WRITE statement. The keyword ##INVOKE has meaning to the FORMS subroutine package and FORMS user to set up a function call to the run-time directive ##INVOKE. The use of the ##INVOKE directive causes the form to be accessed in the FORMS catalog and displayed at the terminal. Data can then be entered into the unprotected fields at the terminal. Use of the ##INVOKE directive is illustrated by the following code:

```
C --- IDENTIFY FORM FOR USE BY APPLICATION PROGRAM,
C --- AND WRITE FORM AT TERMINAL.
C
      WRITE (1,20)
20    FORMAT('##INVOKE ADMN377')
```

This coding identifies the formname to be used as ADMN377. When executed, the code causes FORMS to search for ADMN377 in the forms catalog, which is contained in a UFD named FORMS* . Then, FORMS displays the form at the terminal as previously described.

Reading and Resetting Control Number

This application program accesses a control file named ATS.C to assign a number to identify the item to the data record currently being prepared at the terminal. The next sequential control number must also be assigned for later use in updating the control file. This is accomplished by the following standard I/O calls and calculation:

```
C --- PROCEDURE TO ASSIGN NEXT ATS #.
C
C --- POSITION TO NEXT DATA RECORD.
C
100   CALL PRWF$$(K$POSN+K$PREA,1,LOC(0),0,000000,NWIO,CODE)
C --- READ ATS.C TO GET ATSNUM
C
      READ (5,120,ERR=160,END=160) ATSNUM
120   FORMAT(I6)
      GO TO 200
C
C --- HERE ON EOF, ETC.
C
160   ATSNUM=0
C
C --- ASSIGN NEXT SEQUENTIAL ATS #.
180   ATSNUM=ATSNUM+1
```

Using Substreams

Figure 7-2 shows how a STREAM (i.e., the data record) typically may be subdivided into SUBSTREAMs (i.e., logical records).

The use of substreams is not only a means for setting up logical records within the data record. Normally, FORMS expects the input and output data record(s) to be only one line of information. Substreams provide a way that the user may specify data records greater than normally expected. Substreams each may contain up to a line's worth of characters. As an aggregate, substreams make up a data record (STREAM). This program demonstrates the use of substreams with FORMS. When the user desires to logically separate data into several records, FORMS substreams are used. Each substream is transferred as an individual data record when the application program is performing I/O operations. For example, it is desirable to output any error messages

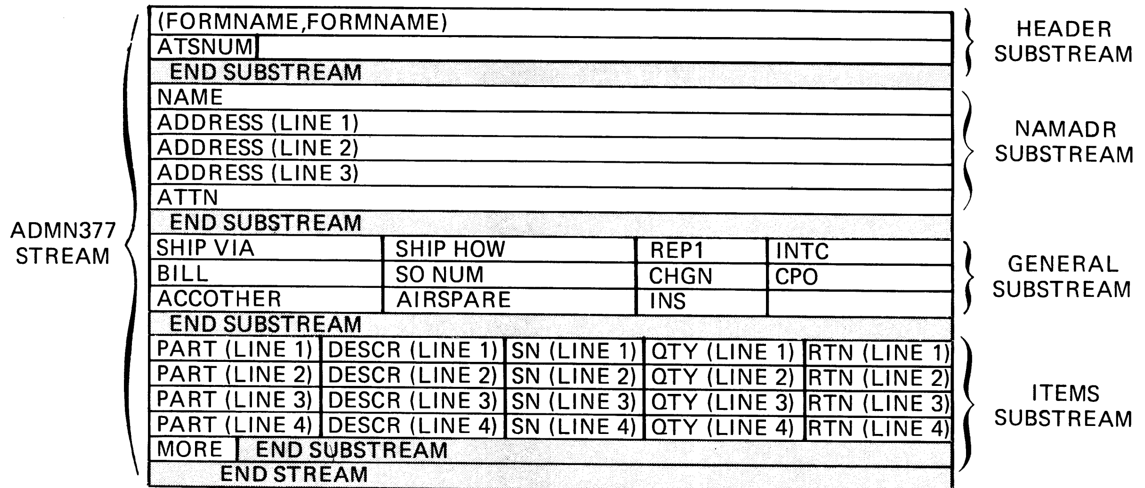


Figure 7-2. SUBSTREAMS Within Data STREAM.

in a separate substream (record). In the sample program, this is done by:

```
C --- WRITE ATS#, CLEAR VARIABLE DATA, ERROR MESSAGE.
C
200   WRITE (1,210) ATSNUM
        HDROUT=.FALSE.    /*HEADER NOT OUTPUT TO DISK FILE
C --- WRITE HEADER AND ERROR MESSAGE FIELD THEN
C --- UNPROTECTED DATA FIELDS,
C --- USING FORMS ##SUBSTREAM AND ##CLEAR DIRECTIVES.
C
210   FORMAT('##SUBSTREAM HEADER'/I6/'##CLEAR'/'##SUBSTREAM ERROR'/'')
```

Statement 210 writes the HEADER and ERROR substreams to the terminal. It also writes the control number and clears the unprotected (variable) data areas on the terminal screen.

Entering Data from the Terminal

The following lines of code read the various data items after the user has entered them into the appropriate fields displayed at the terminal and has pressed the "ENTER" key. Some of the accompanying validation data is shown. For example, if a blank name and address (NAMADR) is entered, the program branches to code that updates the control file, clears the screen, and exits. (See "exiting" later in this section.) FORMS directives are not used during this phase of program operation.

```
C --- READ IN NAMES, ADDRESSES, AND ACCOUNTING INFORMATION.
C
220   READ (1,240) NAMADR, VIA, HOW, REPL, INTC, BILL, SONUM,
     +             CHGNUM, CPO, ACOTHR, AIRSPR, INS
C
      IF (NAMADR(1).EQ.'  ') GO TO 5000    /* BLANK NAME => EXIT
C --- READ NAMADR. NOTE 75 WORDS = 150 CHARACTERS, ALSO
C --- READ ACCOUNTING INFO (VARIABLES SUCH AS: VIA, HOW, ETC.)
240   FORMAT(75A2,2I1,3A1,12A2,15A2,A1,4A2,A1)
```

Writing Data to the Output File

The file header and account header are written to the disk file, followed by individual data items. Standard FORTRAN I/O is used as follows:

```
C --- WRITE DATA TO DISK FILE.
C
      DO 550 I=1,4
      IF (DESCR(1,I).EQ.'  ') GO TO 550   /* IGNORE BLANK LINE
      IF (HDROUT) GO TO 540
C
C --- WRITE ACCOUNT HEADER TO DISK FILE.
C
      WRITE (6,525) ATSNUM, NAMADR, VIA, HOW, REPL, INTC, BILL, SONUM,
     +              CHGNUM, CPO, ACOTHR, AIRSPR, INS
525   FORMAT('*ATS',I6/5(15A2/),2I1,3A1,12A2/15A2/A1,4A2,A1)
C
      HDROUT=.TRUE.
C
C --- WRITE INDIVIDUAL ITEM LINE TO DISK FILE.
C
540   WRITE (6,545) (PART(J,I),J=1,8), (DESCR(J,I),J=1,15),
     +              (SN(J,I),J=1,4), QTY(I), RTN(I)
545   FORMAT(7A2,A1,15A2,4A2,I4,A1)
C
550   CONTINUE
```

Data Checking

The application program checks if there are any more items to be written, by checking for nonzero data in the MORE field displayed at the terminal. If the user has entered data other than zero in the MORE field, it indicates that more information with respect to part numbers etc is to be associated with the current order. Data items are written using the ##SUBSTREAM directive. This is shown in the following code, along with the test for "more" data. As each data item is entered from the terminal and written to the terminal, the terminal cursor is positioned the the next field to be read or written through use of the ##POSITION directive. For example, see statement 580 of the following code.

```
C --- CHECK FOR MORE ITEM LINES.
C
C --- IF NO MORE ITEMS, THEN EXIT.
C
      IF (MORE.EQ.' '.OR.MORE.EQ.'N'.OR.MORE.EQ.'n') GO TO 180
C --- USE FORMS ##SUBSTREAM AND ##POSITION DIRECTIVES TO
C --- WRITE OUTPUT DATA RECORD.
C
C --- DATA ITEMS HAVE BEEN ENTERED AT THE TERMINAL BY USER.
C
C
      WRITE (1,580)
580   FORMAT('##SUBSTREAM ITEMS'/' '/'##SUBSTREAM ITEMS'/
     +       '##POSITION PART01')
      GO TO 400   /* NEXT SET OF ITEM LINES
```

Response to Data Input Errors

Checks are made that the proper range or type of data values are entered by the user at the terminal. If a data input error occurs, the program usually branches to one of a number of internal routines that print and error message in the error substream and position the terminal cursor back to receive a correct value for the data item that was in error. These routines uniformly employ the ##SUBSTREAM directive to print the message at the terminal and the ##POSITION directive to reposition the cursor. This is illustrated by the following typical lines of code:

```
C --- INCORRECT DATA IN ACCOUNTING FIELDS.
C
C ACTION TAKEN ON INCORRECT 'VIA' CODE:
C
1000  WRITE (1,1010)
1010  FORMAT('##SUBSTREAM ERROR'/
     +       'via code must be 1-9'/
     +       '##POSITION SHIPVIA')
      GO TO 220
C
C --- ACTION TAKEN ON INCORRECT 'HOW' CODE:
C
1020  WRITE (1,1030)
1030  FORMAT('##SUBSTREAM ERROR'/
     +       'how code must be 1-4'/
     +       '##POSITION SHIPHOW')
      GO TO 220
C
C --- YES/NO ANSWER REQUIRED.
C
1040  WRITE (1,1050) (FLD1(J,I), J=1,4)
1050  FORMAT('##SUBSTREAM ERROR'/
     +       'yes/no (Y or N) response required'/
     +       '##POSITION ',4A2)
      GO TO 220
```

Exiting

The exit code updates the control file, closes the control file and output file, clears the terminal, and exits to PRIMOS. The exiting code is initiated when the program encounters a NAMADR field that is blank which will occur when the user enters a blank in the NAMADR field. The test statement for a NAMADR of blank as well as the exit code statements are shown in the following excerpts from the application program.

```
C --- HERE TO EXIT.  UPDATE ATS # IN CONTROL FILE.
C
5000  CALL PRWF$$(K$POSN+K$PREA,1,LOC(0),0,000000,NWIO, CODE)
C --- WRITE ATSNUM IN ATS.C .
      WRITE(5,120) ATSNUM
      CALL PRWF$$(K$TRNC,1,LOC(0),0,000000,NWIO,CODE)
C
C --- CLOSE CONTROL FILE AND DATA FILE.
      CALL SRCH$$(K$CLOS,0,0,1,0,CODE)
      CALL SRCH$$(K$CLOS,0,0,2,0,CODE)
C
C --- USE FORM ##CLEAR DIRECTIVE TO CLEAR TERMINAL DATA FIELDS, AND
C --- RELEASE SCREEN AND RETURN IT FOR USE BY PRIMOS.
C
      WRITE (1,5020)
5020  FORMAT('##CLEAR ALL'/'##RELEASE')
C
      CALL EXIT
      END
```

CREATING THE FORM DESCRIPTOR FILE

The foregoing example discussed how the example application program was coded and reasons for using the various coding techniques. In addition, the user must be concerned with describing the form to be used with the program. The data contained with the form, both on the terminal screen and in the data record must be described using the FDL translator and placed on the FORMS catalog using the FAP command (see Section 6). The form descriptor specified by the user as a template that describes both how the form is to be displayed at the terminal and how it is to be stored, in terms of: the attributes of the data items and the overall arrangement of the form.

As stated in Sections 3 and 5, the FDL forms descriptor language is available to make a source language description of what the form looks like both on the terminal and in the data record. The syntax of FDL is summarized in Section 3 and described in Section 5. The following paragraphs discuss how the form used by the sample application program is coded and translated. A complete listing of the sample form description is given in Appendix F. Features that are essential in every form description as well as some special features are discussed in this section together with some excerpts of the FDL source code. Features discussed in some detail are:

- setting up the FORMS descriptors
- defining the terminal devices used
- initializing the data record and terminal screen display
- defining data areas (fields)
- differentiating input data from output data
- differentiating literal data and variable data
- using substreams
- using the macro definition capability of FDL
- specifying listing details

The discussion is completed by a brief description of how to use the FDL language translator command and how the FDL checks for errors in the source language input. This is followed by a discussion of how to install a form description in the FORMS catalog using FAP.

Setting Up a Form Description

Every form description consists of two parts: one to describe the data record, and the other to describe the form itself. These are the STREAM and the FORMAT descriptors respectively. The STREAM descriptor is bounded (defined) by a STREAM statement and an END STREAM statement.

Likewise, a FORMAT descriptor is bounded by a FORMAT and END FORMAT statement. In each form description, there need be only one STREAM descriptor, since the internal data record need be only described once. However, there must be one FORMAT descriptor for each type of device the user wishes to use with the application program. An example of FDL code is given in the following paragraphs.

Defining Device Types

For each terminal connected to the system that interfaces with the application program, there must be a DEVICE descriptor. These descriptors are bounded by DEVICE and END DEVICE and contained within the body of the FORMAT descriptor, as illustrated in the following FDL coding:

```
ADMN377  STREAM
...
         END STREAM
ADMN377  FORMAT
         DEVICE OWL1200
...
         END DEVICE
         DEVICE VISTAR3
...
         END DEVICE
         END FORMAT
```

Using SUBSTREAMS

The discussion on the application program showed how the program took advantage of the ##SUBSTREAM directive and the substream concept of FORMS to effectively use logical records. Within FDL, SUBSTREAM and END SUBSTREAM statements allow a user to define and delimit substreams. The substreams are set up in the STREAM descriptor. The data items (FIELDS) with a SUBSTREAM are mapped to fields in the FORMAT descriptor. In the example FDL coding, the substreams that have been set up correspond to the substreams that are handled by the program, i.e., there are: a header substream, a general substream, a data item substream, and an error substream. The following code excerpt shows how one of the substreams is defined:

```
ADMN377  STREAM
HEADER   SUBSTREAM
         FIELD (FORMNAME,FORMNAME)
         FIELD ATSNUM, LENGTH 6, JUSTIFY RIGHT, ZERO-FILL, OUTPUT
         END SUBSTREAM
*
*
  ...
         END STREAM
```

The following code in the FORMAT descriptor is the mapped equivalent of the sample SUBSTREAM:

```
*--- HEADER LINE INFORMATION:
*
          FIELD 'FORM' POSITION (2,1)
FORMNAME  FIELD LENGTH 8, POSITION (7,1)
          FIELD 'ATS #' POSITION (20,1)
ATSNUM    FIELD LENGTH 6, POSITION (26,1)
*
```

Setting Up Data Areas (FIELDS)

Data items are described by FIELD statements. FIELD statements may describe a literal, some system wide information, or may describe a variable. FIELD statements that describe a literal consist of the keyword FIELD and the literal in single quotes, for example:

```
FIELD 'CUSTOMER NAME'
```

Fields that give system information such as time and date are described in Section 5. FIELD statements that describe variable (NOPROTECT) data items give the length of the field and some of the attributes of the data within the field, such as type, justification, validation, etc. There are differences between the format of a FIELD statement in a STREAM descriptor and a FIELD statement in a FORMAT descriptor. For example, the field name in a STREAM-FIELD statement is contained within the body of the statement following the keyword STREAM, and the field name of a FORMAT descriptor must appear in Column 1. However, there is a one-for-one mapping between the variable fields in a STREAM descriptor and the corresponding fields in the associated FORMAT descriptor. All of these practices are illustrated in the following code:

```
ADMN377  STREAM
  ...
  *--- SHIP TO NAME AND ADDRESS.
  *
  NAMADR  SUBSTREAM
          FIELD NAME, LENGTH 30, VALIDATE 'P' OR 'B'
          REPEAT 3
          FIELD ADDR, LENGTH 30
          END REPEAT
  *
          FIELD ATTN, LENGTH 30
          END SUBSTREAM
  *
   ...
  ADMN377  FORMAT
   ...
  *--- SHIP TO INFORMATION:
  *
            FIELD 'SHIP TO ' POSITION (2,3), RVIDEO
            FIELD 'NAME' POSITION (12,3)
  NAME         FIELD LEN 30, POSITION (24,3), NOPROTECT
            FIELD 'ADDRESS' POSITION (12,4)
            REPEAT 3
  ADDR         FIELD LEN 30, POSITION (24,+3), NOPROTECT
            END REPEAT
            FIELD 'ATTENTION' POSITION (12,7)
  ATTN         FIELD LEN 30 POSITION (24,7) NOPROTECT
  *
   ...
```

Input and Output Data

The listing generated by FDL separates the STREAM description into two areas, one showing INPUT field data and one showing OUTPUT field data. Refer to the listing in Appendix F. If the user does not specify otherwise, all fields are both INPUT/OUTPUT. An example of an output field is illustrated by the following line of code:

```
FIELD ATSNUM, LENGTH 6, JUSTIFY RIGHT, ZERO-FILL, OUTPUT
```

The listing in Appendix F shows how FDL separates INPUT and OUTPUT fields. A check of this listing will reveal that the field ATSNUM is present in the output description but not the input description.

The variable (ATSNUM) is not specified at the terminal by the user, it is generated and used internally by the program.

Literal and Variable Data

Literal fields within a FORMAT are simply specified by the FIELD statement containing the literal string. For example:

```
FIELD 'FORM' POSITION (2,1)
```

Variable fields may be specified by using the NOPROTECT attribute. For example:

```
NAME      FIELD LENGTH 30, POSITION (24,3), NOPROTECT
```

PROTECTed fields may not be written into by the user. A NOPROTECTed field may be written into by the user. For example:

```
ATSNUM    FIELD LENGTH 6, POSITION (26,1), PROTECT
```

Using the Macro Definitions

To show the flexibility and convenience of coding in FDL, the DEFINE statement is used in the sample coding to make up a series of macro definitions that allow abbreviation of key words or often used names. These definitions are:

```
*    MACRO DEFINITIONS FOR FORMS DEFINITION LANGUAGE TRANSLATOR
*    COPYRIGHT 1979, PRIME COMPUTER, FRAMINGHAM MA
*
F         DEF FIELD
V         DEF VALIDATE
LEN       DEF LENGTH
POS       DEF POSITION
IN        DEF INPUT
OUT       DEF OUTPUT
JUS       DEF JUSTIFY
R         DEF RIGHT
L         DEF LEFT
C         DEF CENTER
NP        DEF NOPROTECT
RV        DEF REVERSE VIDEO
BL        DEF BLINK
*
*---END FORMS>MACROS
```

Heretofore, example FDL code in this section were all spelled out. However, as Appendix F shows, the user may to take advantage of the abbreviation macros. For example consider the line:

```
F NAME, LEN 30, V 'P' OR 'B'
```

as opposed to:

```
FIELD NAME, LENGTH 30, VALIDATE 'P' OR 'B'
```

This saves time in typing input when often-used words such as VALIDATE are abbreviated to one letter (i.e., V).

CAUTION

> Using abbreviations makes it difficult or impossible to maintain the application program and its associated form descriptors.

Macros may be written in-line -- in the FDL code as shown here or may be inserted by the FDL statement $INSERT. The depth of a macro definition is internal to the descriptor that contains it. Thus, in the example FDL listing in Appendix F, the macro definition statements in the STREAM are in-line. In the FORMAT, the same macro definitions are referenced by a $INSERT statement.

Specifying FDL Listing details

FDL produces an output listing unless the user specifies otherwise. The user can control listing features with control statements such as $LIST and $NOLIST. One advantage of the listing is that the user can check and be sure his data item in the STREAM portion of the description map properly to the corresponding data item the FORMAT portion. For example:

```
*--- ITEM INFORMATION.
*
     REPEAT 4
     FIELD PART, LENGTH 15, JUSTIFY RIGHT, SPACE-FILL
     FIELD DESCR, LENGTH 30, JUSTIFY LEFT
     FIELD SN, LENGTH 8, JUSTIFY RIGHT, ZERO-FILL
     FIELD QTY, LENGTH 4, JUSTIFY RIGHT, ZERO-FILL, VALIDATE '9' OR 'B'
     FIELD RTN, LENGTH 1, VALIDATE 'A' OR 'B'
     END REPEAT
```

in the SUBSTREAM fields correspond (i.e., map) to the following items in the FORMAT fields:

```
           REPEAT 4
PART       FIELD LENGTH 15, POSITION (2,+18), NOPROTECT
DESCR      FIELD LENGTH 30, POSITION (19,+18), NOPROTECT
SN         FIELD LENGTH 8, POSITION (51,+18), NOPROTECT
QTY        FIELD LENGTH 4, POSITION (61,+18), NOPROTECT
RTN        FIELD LENGTH 1, POSITION (67,+18), NOPROTECT
           END REPEAT
```

Repeated Text

The code in the previous example shows the use of the REPEAT feature in FDL which allows a user to specify the duplication of similar fields over several lines on the terminal screen. In this case, the item "PARTS" was repeated four times using REPEAT 4. The END REPEAT statement delimits the domain of a REPEAT.

Listing Features

Finally, the FDL output listing shows some additional useful information about the form descriptor. First, as shown in Appendix F, the items defined in the input stream and the output stream are listed in a separate cross-index after the listing of the translated STREAM code. For example, the header substream is OUTPUT but not INPUT information. Second, the listing provides a convenient chart of lines and columns for each DEVICE specified in the FORMAT descriptor(See Appendix F and Appendix H). For example, the last two pages of the listing in Appendix F of the output listing (named ADMN377) show the fields in their relative position location, show literal fields as they appear, and show the location of the variable (NOPROTECTed) fields by the lines of dashes. Refer to the listing in Appendix F for further information.

COMPILING THE APPLICATION PROGRAM

The example program is compiled using the FTN command as follows:

```
OK, FTN MAIN -64V -LIST
GO
0000 ERRORS [<.MAIN.>FTN-REV15.3]

OK,
```

TRANSLATING THE FDL SOURCE

The source description of the form must be input to the FDL translator, using the FDL command. The translation produces an output file in a binary form which may be placed in the FORMS catalog. Files listed in the FORMS catalog can be read and interpreted by the application program at run-time. The following is an example of the FDL translation for the sample form descriptor named ADMN377.

```
OK, FDL ADMN377
GO
0000 ERRORS (FDL, REV 16 - 16-FEB-79)
0000 ERRORS (FDL, REV 16 - 16-FEB-79)

OK,
```

INSTALLING FORM DESCRIPTOR IN FORMS CATALOG

The FAP command is used to install forms descriptors in the FORMS catalog and otherwise maintain this library. The following is a sample installation of the form descriptor ADMN377 in the FORMS catalog:

```
OK, FAP
GO

FAP REV 16   23-DEC-78

* ADD B ADMN377
  3 DEFINITIONS ADDED
* QUIT

OK,
```

LOADING THE APPLICATION PROGRAM

Since the sample program is compiled in V-mode the SEG loader must be used, and the appropriate V-mode libraries must be invoked. This procedure is illustrated in the following example:

```
OK, SEG
GO
# LOAD
SAVE FILE TREE NAME: #MAIN
$ LO B_MAIN
$ LIB VFORMS
$ LIB VSPOO$
$ LIB
LOAD COMPLETE
$ SAVE
$ QUIT

OK,
```

RUNNING THE PROGRAM

The program may be run by a simple invocation of the SEG command, as follows:

```
SEG #MAIN
```

SECTION 8

EXAMPLE COBOL PROGRAM

INTRODUCTION

This section describes the development of a FORMS application program from source coding to loading and execution. One of the significant differences between the program described in this section and the one described in Section 7, besides its being written in COBOL, is the fact that the program uses keyed-index files and interfaces with MIDAS as well as with FORMS. This section also describes the data used with the application program that defines the form and describes how to place the form descriptor in the FORMS catalog.

The example program (named DEM01) accepts a simple list of orders, typed at the terminal, and writes the items on the order list to a keyed-index file (named DS1). The data in the form is handled as a simple STREAM (i.e., only one logical record). An example of the screen, observed by the user, is shown in Figure 8-1. Features that are essential to every FORMS application program, as well as some special features unique to the application are discussed with source code excerpts. Features discussed in this section are:

- Defining files.
- Setting up data areas through use of standard COBOL statements.
- Defining data screen.
- Initializing the descriptor for the form to be used to means of a COBOL read statement that transfers the FORMS directive ##INVOKE to the terminal device driver and causes the form to be displayed on the terminal screen.
- Writing the header and other data to the terminal using FORMS directives.
- Accepting input data from the terminal using READ and FORMS run-time directives.
- Positioning the terminal cursor to the home position and clearing the terminal screen of the last block of data written.
- Error handling by the application program.

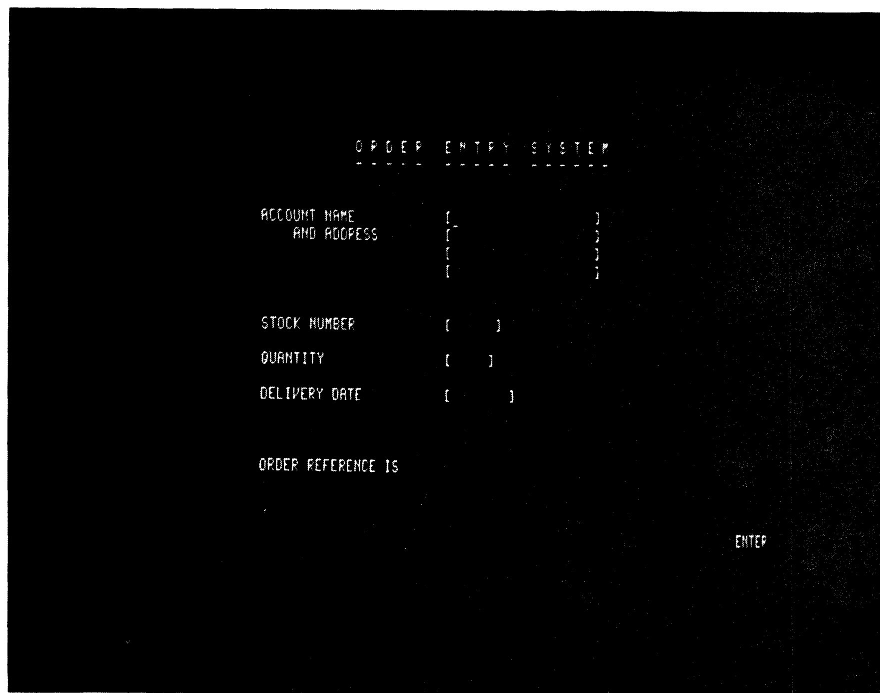


Figure 8-1. Screen Display Produced by Running Example Program.

- Exiting from the application program.

WRITING THE PROGRAM

A complete listing of the program's source coding appears in Appendix G.

Defining Files

The files to be used are defined with a standard COBOL programming practice, i.e., by the FILE CONTROL SECTION as follows:

```
FILE-CONTROL.
    SELECT INFILE ASSIGN TO TERMINAL.
    SELECT ORDER-FILE ASSIGN TO PFMS
      ORGANIZATION IS INDEXED
      ACCESS IS DYNAMIC
      RECORD KEY IS ORDER-KEY
      ALTERNATE RECORD KEY IS ORDER-NAME WITH DUPLICATES
      ALTERNATE RECORD KEY IS STOCK-NO WITH DUPLICATES
      ALTERNATE RECORD KEY IS STOCK-DEL WITH DUPLICATES.
```

Setting-Up Data Areas

The example program sets up the data areas, first, by defining the input and output data records in the FILE SECTION of the DATA DIVISION. This is shown in the following code:

```
        DATA DIVISION.
        FILE SECTION.
        FD  INFILE  LABEL RECORDS ARE OMITTED.
        01  SCR.
            02  FILLER          PIC X(104).
        FD ORDER-FILE LABEL RECORDS ARE STANDARD
                    VALUE OF FILE-ID IS "ORDERS".
        01 ORDER-RECORD.
            02 ORDER-KEY.
                03  ORDER-NO             PIC 9(5).
                03  ORDER-ITEM           PIC 99.
            02 ORDER-NAME       PIC X(20).
            02 ORDER-ADD1       PIC X(20).
            02 ORDER-ADD2       PIC X(20).
            02 ORDER-ADD3       PIC X(20).
            02 STOCK-NO         PIC X(6).
            02 STOCK-DEL        PIC X(8).
            02 STOCK-QTY        PIC S9(5) SIGN TRAILING SEPARATE.
       01  ORDERC.
           02 FILLER            PIC X(7).
           02 NEXT-ORDER        PIC 9(5).
           02 FILLER            PIC X(94).
```

Extending Size of Terminal I/O Buffers

The number of characters handled by a single transaction by this program (150) happens to be more than the default line size of the terminal read-write buffers (72). However, the size of the I/O terminal buffers may be changed by calls to the Primos subroutine ATTDEV. For example:

```
      * EXTEND TERMINAL, FILE I/O BUFFERS.
      *
           MOVE 0 TO VAR0.
           MOVE 1 TO VAR1.
           MOVE 2 TO VAR2.
           MOVE 6 TO VAR6.
           MOVE 7 TO VAR7.
           MOVE 150 TO VAR150.
           CALL 'ATTDEV' USING VAR1, VAR1, VAR0, VAR150.
           CALL 'ATTDEV' USING VAR6, VAR7, VAR2, VAR150.
```

The first call to ATTDEV sets up an input buffer, open for reading, from the terminal on File Unit 1. The second call to ATTDEV sets up an output buffer, open for writing on File Unit 6. In both cases, the size of the buffer is specified to be 150 characters.

Shared Library Initialization

In order to use the FORMS run-time directives with the shared libraries, the following call must be present in the application program prior to the first ##INVOKE directive:

```
CALL 'FORM$I'.
```

This requirement is applicable to the 64V mode shared version of FORMS and is ignored for 64R mode and non-shared 64V mode.

Selecting a Library

It is a system administration decision whether or not to support the shared libraries. If shared FORMS is supported, shared COBOL, MIDAS, and FORTRAN must be supported as well. If shared libraries are in use, the FORMS shared library file is named VFORMS.

Defining FORMS Directives

FORM directives used by the application program are defined in the WORKING STORAGE SECTION of the DATA DIVISION. This is one of the areas in which the sample program differs from an "ordinary" COBOL program. The definition of the FORM directives to be used appears as follows:

```
 WORKING-STORAGE SECTION.
* DEFINITION OF FORMS DIRECTIVES
 77  INV-C   PIC X(12)  VALUE '##INVOKE DS1'.
 77  REL-C    PIC X(09) VALUE '##RELEASE'.
 77  PROT-C  PIC X(25) VALUE '##PROTECT NAM AD1 AD2 AD3'.
 77  ENAB-C  PIC X(24) VALUE '##ENABLE NAM AD1 AD2 AD3'.
 77  CLEAR-C  PIC X(07) VALUE '##CLEAR'.
 77  CLA-C    PIC X(11) VALUE '##CLEAR ALL'.
* CURRENT ORDER
 77  CURR-ORD     PIC 9(5).
```

Defining Terminal Screen

Also in the WORKING STORAGE SECTION, the program defines the individual data items that appear on the terminal screen (DATA-SCREEN), as follows:

```
01  DATA-SCREEN.
* CUSTOMER AND ORDER INFORMATION
    02  DS-NAM.
        03  CUSIND      PIC XX.
        03  FILLER      PIC X(18).
    02  DS-AD1       PIC X(20).
    02  DS-AD2       PIC X(20).
    02  DS-AD3       PIC X(20).
    02  DS-STK.
        03  STK-ID      PIC XX.
        03  FILLER      PIC X(4).
    02  DS-QTY       PIC 9(5).
    02  DS-DATE      PIC X(8).
    02  DS-ORDER     PIC 9(5).
```

The items defined as part of the form will be moved to the order record defined previously in the FILE SECTION.

Opening Files

One of the first steps in the procedure is to open the input and output files, as shown by the following:

```
PROCEDURE DIVISION.
START-POINT.
    OPEN I-O ORDER-FILE.
    OPEN I-O INFILE.
```

Invoking Form

INV-C has been defined as the FORMS ##INVOKE directive, thus

```
WRITE SCR FROM INV-C
```

causes the form to be displayed.

Reading and Checking Input

After the data is entered at the terminal by the user, the program reads this data by the following code:

```
B-POINT.
   READ INFILE INTO DATA-SCREEN.
   IF CUSIND EQUAL '**' GO TO END-IT.
```

If the user has input a double asterisk (**) for the variable CUSIND, then the program exits to PRIMOS command level.

Writing Data

The following code writes any existing data to the output file and returns, ready to receive additional input from the user terminal:

```
 TRY-IT.
     MOVE ZEROES TO ORDER-KEY.
     READ ORDER-FILE KEY IS ORDER-KEY INVALID KEY
          GO TO TRY-IT.
     ADD 1 TO NEXT-ORDER.
     MOVE NEXT-ORDER TO CURR-ORD.
     REWRITE ORDER-RECORD.
*
     MOVE DS-NAM TO ORDER-NAME.
     MOVE DS-AD1 TO ORDER-ADD1.
     MOVE DS-AD2 TO ORDER-ADD2.
     MOVE DS-AD3 TO ORDER-ADD3.
     MOVE ZERO TO ORDER-ITEM.
 A-POINT.
     MOVE DS-STK TO STOCK-NO.
     INSPECT DS-STK REPLACING LEADING SPACES BY ZEROES.
     MOVE DS-QTY TO STOCK-QTY.
     MOVE DS-DATE TO STOCK-DEL.
*
     ADD 1 TO ORDER-ITEM.
     MOVE CURR-ORD TO ORDER-NO.
     MOVE CURR-ORD TO DS-ORDER.
*
     WRITE ORDER-RECORD.
     WRITE SCR FROM PROT-C.
     MOVE SPACES TO DS-STK, DS-QTY, DS-DATE.
     WRITE SCR FROM DATA-SCREEN.
     READ INFILE INTO DATA-SCREEN.
     IF STK-ID NOT EQUAL '**' GO TO A-POINT.
*
     WRITE SCR FROM ENAB-C.
     WRITE SCR FROM CLEAR-C.
     GO TO B-POINT.
```

Exiting

FORMS directives exist to clear and release the terminal screen. They are called when the program exits, by invoking the working storage variables CLA-C and REL-C. The exit code is as follows:

```
END-IT.
    CLOSE ORDER-FILE.
    WRITE SCR FROM CLA-C.
    WRITE SCR FROM REL-C.
    CLOSE INFILE.
    STOP 'END OF ORDER ENTRY'.
```

CREATING THE FORM DESCRIPTOR FILE

The foregoing example shows how an application program might be coded and discussed reasons for using various coding techniques. In addition, the user must describe the form to be used with the program. The data contained with the form, both on the terminal screen and in the data record must be described using the FDL translator and placed on the FORMS library (called the FORMS catalog) using the FAP command (see Section 6). The form descriptor specified by the user as a template that describes both how the form is to be displayed at the terminal and how it is to be stored, in terms of the attributes of the data items and the overall arrangement of the form.

As stated in Sections 3 and 5, the FDL translator forms descriptor language is available to make a source language description of what the form looks like both on the terminal and in the data record. The syntax of FDL is summarized in Section 3 and described in Section 5. The following paragraphs discuss how the form used by the sample application program is coded and tanslated. A complete listing of the sample form description is given in Appendix H. Features that are essential in every form description as well as special features are discussed in this section together with excerpts of the FDL source code. Features discussed in detail are:

- Setting up the FORMS descriptors
- Defining the terminal devices used
- Defining data areas (fields)
- Differentiating input data from output data
- Differentiating literal data and variable data
- Specifying listing details

The discussion is completed by a brief description of how to use the FDL language translator command and how the FDL checks for errors in the source language input. This is followed by a discussion of how to install a form description in the FORMS catalog using FAP.

Setting Up Form Description

Every form description consists of two parts: one to describe the data record, and the other to describe the form itself. These are the STREAM and the FORMAT descriptors respectively. The STREAM descriptor is bounded (defined) by a STREAM statement and an END STREAM statement. Likewise, a FORMAT descriptor is bounded by a FORMAT and END FORMAT statement. In each form description, there need by only one STREAM descriptor, since the internal data record need be only described once. However, there must be one FORMAT descriptor for each type of device the user wishes to use with the application program. An example of FDL code is shown in the following paragraphs.

Defining Device Types

For each terminal connected to the system that interfaces with the application program, there must be a DEVICE descriptor. These descriptors are bounded by DEVICE and END DEVICE statements and are contained within the body of the FORMAT dscriptor, as illustrated in the following FDL coding excerpts:

```
DS1     STREAM
..
        END STREAM
DS1     FORMAT
        DEVICE OWL1200
..
        END DEVICE
        DEVICE VISTAR3
..
        END DEVICE
        END FORMAT
```

Setting Up Data Areas (FIELDS)

Data items are described by FIELD statements. FIELD statements may describe a literal, some system wide information, or may describe a variable. FIELD statements that describe a literal consist of the keyword FIELD and the literal in single quotes, for example:

```
FIELD 'CUSTOMER NAME'
```

Fields that give system information such as time and date are described in Section 5. FIELD statements that describe variable (NOPROTECT) data items give the length of the field and the attributes of the data within the field, such as type, justification, validation, etc. There are differences between the format of a FIELD statement in a STREAM descriptor and a FIELD statement in a FORMAT descriptor. For example, the field name in a STREAM-FIELD statement is contained within the body of the statement following the keyword STREAM, and the field name of a FORMAT-FIELD statement must appear in Column 1. However, there is a one-for-one mapping between the variable fields in a STREAM descriptor and the corresponding fields in the associated FORMAT descriptor. All of these practices are illustrated in the following code:

```
* DATA RECORD LAYOUT
*
S1    STREAM
* ACCOUNT NAME
*
AM    FIELD NAM LENGTH 20
* ADDRESS
*
D1    FIELD AD1 LENGTH 20
D2    FIELD AD2 LENGTH 20
D3    FIELD AD3 LENGTH 20
* STOCK - ORDER INFORMATION
*
TKNNO FIELD STKNO LENGTH 6
TKQTY FIELD STKQTY LENGTH 5 JUSTIFY RIGHT
TKDEL FIELD STKDEL LENGTH 8
* ORDER NUMBER - GENERATED BY PROGRAM AND OUTPUT
*
NO    FIELD ONO LENGTH 5 OUTPUT
      END STREAM
* END OF DATA RECORD DESCRIPTION
**********
* FORM DESCRIPTION
S1    FORMAT
* FIRST DEVICE DESCRIPTION
*
      DEVICE VISTAR3
* HEADING INFORMATION OF FORM
*
      FIELD 'O R D E R  E N T R Y  S Y S T E M' POSITION (22,2)
      FIELD '- - - - -  - - - - -  - - - - - -' POSITION (22,3)
      FIELD 'ACCOUNT NAME     ' POSITION (10,6)
* ACCOUNT NAME INFORMATION
AM    FIELD LENGTH 20 POSITION (35,6) NOPROTECT
      FIELD '    AND ADDRESS' POSITION (10,7)
* ACCOUNT ADDRESS
*
D1    FIELD LENGTH 20 POSITION (35,7) NOPROTECT
D2    FIELD LENGTH 20 POSITION (35,8) NOPROTECT
D3    FIELD LENGTH 20 POSITION (35,9) NOPROTECT
      FIELD 'STOCK NUMBER' POSITION (10,12)
* STOCK - ORDER INFORMATION
*
TKNO  FIELD LENGTH 6 POSITION (35,12) NOPROTECT
      FIELD 'QUANTITY' POSITION (10,14)
TKQTY FIELD LENGTH 5 POSITION (35,14) NOPROTECT
      FIELD 'DELIVERY DATE' POSITION (10,16)
TKDEL FIELD LENGTH 8 POSITION (35,16) NOPROTECT
      FIELD 'ORDER REFERENCE IS' POSITION (10,20)
NO    FIELD LENGTH 5 POSITION (35,20)
      END DEVICE
* END OF FIRST DEVICE DESCRIPTION
     ...
```

In this case, mapping has been performed by the programmer, by giving the STREAM fields the same label as the corresponding FORMAT fields. However, this is not necessary if the STREAM variables are given the same name as the FORMAT labels.

Input and Output Data

The listing generated by FDL separates the STREAM description into two areas, one showing INPUT field data and one showing OUTPUT field data. Refer to the listing in Appendix H. If the user does not specify otherwise, all fields are both INPUT/OUTPUT. An example of an OUTPUT field is illustrated by the following line of code:

```
NO FIELD ONO, LENGTH 5, OUTPUT
```

The listing in Appendix H shows how FDL separates INPUT and OUTPUT fields. A check of this listing will reveal that the field ONO is present in the output description but not the input description.

The variable (ONO) is not specified at the terminal by the user, it is generated and used internally by the program. Therefore, there is no reason for it to be specified as input, it only need be written on output.

Literal and Variable Data

Literal fields within a FORMAT are simply specified by the FIELD statement containing the literal string. For example:

```
FIELD 'ACCOUNT NAME ' POSITION (10,6)
```

Variable fields may be specified by using the NOPROTECT attribute. For example:

```
FIELD LENGTH 30, POSITION (35,6), NOPROTECT
```

Fields that may not be changed may prevent the user from writing in them by use of the PROTECT attribute. (Refer to Section 5.)

Specifying FDL Listing details

FDL produces an output listing unless the user specifies otherwise. The user can control listing features with control statements such as LIST and NOLIST. One advantage of the listing is that the user can check and be sure the data items in the STREAM portion of the description map properly to the corresponding data items the FORMAT portion.

Finally, the FDL output listing shows additional useful information about the form description. First the items defined in the input stream and the output stream are listed in a separate cross-index after the listing of the translated STREAM code. For example, the header substream is output but not input information. Similarly, the listing

provides a convenient chart of lines and columns for each DEVICE specified in the FORMAT descriptor. For example, the last two pages of the output listing (named DS1) in Appendix H show the fields in their relative position location, show literal fields as they appear, and show the location of the variable (NOPROTECTED) fields by the lines of dashes.

COMPILING THE APPLICATION PROGRAM

The example program is compiled using the COBOL command, as follows:

```
OK, COBOL DEMO1 -64V -LIST
GO
 PHASE I
 PHASE II
 PHASE III
 PHASE IV
 PHASE V
 PHASE VI


NO ERRORS,  1 WARNINGS,  P400/500 COBOL REV 15.3  <MAIN  >

OK,
```

TRANSLATING THE FDL SOURCE

The source description of the form must be input to the FDL translator, using the FDL command. The translation produces an output file in a binary form which may be placed in the FORMS catalog. Files listed in the FORMS catalog can be read and interpreted by the application program at run-time. The following is an example of the FDL translation for the sample form descriptor named DS1.

```
OK, FDL DS1
GO
0000 ERRORS (FDL, REV 16 - 16-FEB-79)
0000 ERRORS (FDL, REV 16 - 16-FEB-79)

OK,
```

INSTALLING FORM DESCRIPTOR IN FORMS LIBRARY

The FAP command is used to install forms descriptors in the FORMS library and also maintain the FORMS catalog. The following is an example installation of the form descriptor DS1 in the FORMS catalog:

```
OK, FAP
GO

FAP REV 16   23-DEC-78

* ADD B_DS1
  3 DEFINITIONS ADDED
* QUIT

OK,
```

LOADING THE APPLICATION PROGRAM

Since the sample program is written and compiled in V-mode the SEG loader must be used and the appropriate libraries invoked. This procedure is illustrated in the following example:

```
OK, SEG
GO
# LOAD
SAVE FILE TREE NAME: #DEMO1
$ LO B_DEMO1
$ LIB VCOBLB            /* V-MODE COBOL LIBRARY */
$ LIB VKDALB            /* V-MODE MIDAS LIBRARY */
$ LIB VFORMS            /* V-MODE FORMS LIBRARY AND SHARED LIBRARY */
$ LIB VSPOO$            /* OFF-LINE PRINTING LIBRARY */
$ LIB                   /* STANDARD SUBROUTINE LIBRARY */
LOAD COMPLETE
$ SAVE
$ QUIT

OK,
```

MIDAS FILE TEMPLATE

Since the example program, DEMO1, uses the INDEXED feature of COBOL, the Prime indexed file system, MIDAS, is automatically invoked. (That is why the library VKDALB was loaded.) Since MIDAS (indexed) files are used, it is necessary to create a template using CREATK. The following code shows this procedure. For further information, refer to the MIDAS Reference Guide.

```
OK, CREATK
GO
MINIMUM OPTIONS? YES

FILE NAME? ORDERS
NEW FILE? YES
DIRECT ACCESS? NO

DATA SUBFILE QUESTIONS

KEY TYPE: A
KEY SIZE = : B 7
DATA SIZE = : 53

SECONDARY INDEX

INDEX NO.? 1

DUPLICATE KEYS PERMITTED? YES
KEY TYPE: A
KEY SIZE = : B 20
USER DATA SIZE = : 0


INDEX NO.? 2

DUPLICATE KEYS PERMITTED? YES
KEY TYPE: A
KEY SIZE = : B 6
USER DATA SIZE = : 0


INDEX NO.? 3

DUPLICATE KEYS PERMITTED? YES
KEY TYPE: A
KEY SIZE = : B 8
USER DATA SIZE = : 0


INDEX NO.? 0

OK,
```

RUNNING THE PROGRAM

The program may be run by a simple invocation of the SEG command, for example:

```
SEG #DEM01
```

APPENDIX A

INSTALLATION

DIRECTORY INFORMATION

The Forms Management System is supplied in a single directory on the master disk. Within this directory, named FORMS*, are subdirectories that contain source text for the components of the system and contains command files that are used to build the FORMS system.

The files in the UFD named FORMS* are:

Name	Type	File Description
FDL	subUFD	sources for FDL translator
FAP	subUFD	sources for FAP utility
RUN	subUFD	sources for run-time package
IOS	subUFD	sources for IOCS interface, device drivers
DOC	subUFD	source for design specification
FORMS*	subUFD	skeleton FORMS catalog, system files
C_RLIB	command	creates 64R library from individual object files
C_VLIB	command	cretess 64V library from individual object files
RFORMS	object-file	64R mode FORMS run-time system
VFORMS	object-file	64V mode FORMS run-time system
MACROS	insert-file	$INSERT file containing sample FDL macro definitions
C_INST	command	installs new FORMS system
C_Rnn	comand	upgrades current FORMS system to Rev. nn (where nn is the current revision number.)

INSTALLING A NEW VERSION OF FORMS

To install the current version of FORMS on a system on which there is no existing copy (i.e., to create a new FORMS installation):

1) Use FUTIL to copy the FORMS* subUFD from the FORMS UFD to the MFD on any started up local disk, thus making FORMS* a first-level directory. FORMS will not work properly if the FORMS* directory resides on a disk on a remote system accessed via PRIMENET.

2) Execute the C_INST command file in the FORMS UFD to copy the FAP and FDL programs to CMDNC0 and to copy the RFORMS and VFORMS object files to the library UFD, LIB. The FORMS system is now ready for use.

The following is an example run; user input is underlined.

```
 OK, MAGRST
    READY TO RESTORE:  PARTIAL
    TREENAME:  MFD>FORMS*

    RESTORE COMPLETE

OK, FUTIL
 >FROM FORMS*
 >TO MFD XXXXXX 0
 >TRECPY FORMS
 >FROM FORMS > FAP
 >TO CMDNC0
 >COPY *FAP > FAP
 >FROM FORMS > FDL
 >COPY *FDL > FDL
 >FROMS FORMS*
 >TO LIB
 >COPY RFORMS, VFORMS, SFORMS
 >QUIT
```

UPGRADING A CURRENT INSTALLATION

To upgrade an existing FORMS installation:

1) Execute the C_INST command file to copy the new run files to CMDNC0 and libraries to LIB.

2) Execute the C_Rnn command file. This will create the necessary files in the FORMS* directory to support Rev. nn.

This procedure will not affect existing programs. The user is encouraged, however, to reload his programs with the new libraries to take advantage of the improvements and additional features offered. The following is an example run; user input is underlined.

```
 OK, MAGRST
    READY TO RESTORE:  PARTIAL
    TREENAME:  MFD>FORMS*
    RESTORE COMPLETE

 OK, CO C_INST
    ...
 OK, CO C_RUN
```

Note

The existing customer has the option of replacing the entire FORMS* UFD using the procedure for a new FORMS customer (previously described) and then adding all the current FORM definitions (from scratch). Speed improvement and size reduction provided by a new library will only come to the user that reloads the application program with the new library.

REBUILDING FORMS

The following command files are available to rebuild all or part of the Forms Management System:

Treename	Description
FORMS> FDL> C_SUBS	compile FDL subroutines
C_FDL	compile FDL main, load and save
C_LOAD	Load and save FDL
FORMS> FAP> C_SUB	compile FAP subroutines
C_FAP	compile, load and save FAP
C_LOAD	load and save FAP
FORMS> RUN> C_FMR	compile 64R mode FORMS run-time package
C_FMV	compile 64V mode FORMS run-time package
FORMS >IOS> C_IOR	compile 64R mode I/O system
C_IOV	compile 64V mode I/O system
FORMS> C_RLIB	build 64R mode library from objects
C_VLIB	build 64V mode library from objects
C_BLD	build entire FORMS system.

All command files described above generate no listing of the compiled source. For each command file which compiles source text, exists a corresponding command file, which in addition to generating a binary (object) file, also produces a compilation listing. The names of these command files may be determined by concatenating the standard command file name with the character 'L'. For example, to compile the 64R mode version of the run-time package and generate a listing, one would run the command file C_FMRL instead of C_FMR.

APPENDIX B

DEVICE I-O

DEVICE INPUT/OUTPUT SYSTEM

This appendix describes the layout and operation of the device input/output system. It is most useful to those desiring knowledge of how FORMs works internally. Users who must write their own device drivers will find information about writing device drivers in this appendix.

The device (Input/Output) system is logically associated with the run-time package and consists of two parts. The first is an IOCS interlude to route all terminal and line printer I/O requests through FORMS. The second part performs all device mapping and input/output with any formatted device.

IOCS INTERLUDE

The IOCS interlude interfaces the Prime Input/Output Control System (IOCS) - refer to the Subroutine Reference Guide, Software Library - to FORMS through a series of input/output subroutines. Included in the IOCS interlude, are replacements for the standard read and write ASCII tables (RATBL and WATBL.) These tables cause FORMS subroutines I$FM01, O$FM01, and O$FM06 to be called to process terminal input, terminal output, and line printer output, respectively. Input and output is processed by I$FM01, O$FM01 and O$FM06 as described in the following paragraphs.

- If the first two character positions in an output record contain two hash marks (#), the output record is passed to the FORMS directive interpreter (FM$CMD).

- If no form is invoked on the associated device, the FORMS subroutine calls the standard IOCS subroutine for that device. O$AA01 is used for terminal output, I$AA12 for input. Line printer output is ignored, which is standard procedure in PRIMOS; another method is used to write files to the spooler line printer.

- If a form is in use on the associated device, the input/output request is transferred to the FORMS subroutine FM$IN (input or FM$OUT output.)

DEVICE I/O MECHANISM

The device I/O mechanism interfaces the body of the FORMS run-time package to the formatted device(s) that are in use. This is

accomplished through a mapping scheme, and a collection of device driver subroutines, one for each device supported in the installation.

Device Definition Database

Two files exist in the FORMS UFD (FORMS*) that describe the device configuration of the installation: the device control file and the terminal configuration file.

Device Control File: The device control file (called DCF.AS) describes each device in terms of a unique logical device number (ldn), device name, device driver subroutine name, and page capacity in lines and columns.

The logical device number (not to be confused with an IOCS logical unit number, used in conjunction with the device driver subroutine name) is used to determine the execution time subroutine address for the device driver for a given device name.

The device control file consists of a series of entries. The format for the device control file entries, which may be modified with the text editor, is as follows:

> Each entry describes one device. It consists of a single line of text with five items, each separated by commas as follows:
>
> ldn, device-name, device-driver-name, lines, columns
>
> ldn is the unique logical device number associated with the device. It must be in the range 1<=ldn>=99. Logical devices 1 to 9 are reserved for use by Prime. User-written device drivers may use any ldn greater than 9.
>
> device-name is the 1 to 8 character device name. It must conform of the naming conventions set forth in the FDL description.
>
> device-driver-name is a two character abbreviation of the name of the device driver subroutine. The full name of the device driver subroutine is xx$IO; xx is the two-character abbreviation.
>
> lines, columns defines the physical device page size.

For example, the contents of the current device control file is:

```
1,PRINTER,PR,66,132
3,VISTAR,V3,24,80
4,OWL1200,OW,24,80
```

Terminal Configuration File: Another file exists that describes the device name for each FORMS terminal on the system. Each user on the PRIMOS system is assigned a unique user number based on the physical

line to which the terminal is connected. The terminal configuration file, (TCB.BN), specifies the terminal name for each of the up 64 FORMS user (terminals) connected to the system. The FAP utility is used to modify and inspect this file.

Device Mapping Scheme

The following paragraphs describe the mapping scheme used by the run-time package.

The FORMS run-time packages has a section of code to perform all initialization. Among other functions, this obtains the device names for the terminal and printer.

The terminal device name is obtained from the Terminal Configuration File, based on the user number (assigned by PRIMOS). The printer name is specified in FORMS as PRINTER.

When the device name is known, the logical device number is otained from the Device Control File. Two versions of the DCF exist. DCF.AS is the ASCII (edit) version that may be changed at any time by the user. DCF.BN is the binary version that is used by the run-time package. DCF.BN file is generated by FAP upon execution of the GENERATE command. Two versions of the file exist to ensure the active copy (DCF.BN) is concurrent with the device address tables.

The logical device number is retained by FORMS and used to identify the device to the device interlude subroutine. This routine dispatches to appropriate device driver using a supplied unique locial device number (LDN). The dispatch operation is performed using device address tables. These tables are generated by FAP (using the GENERATE command) and compiled into the I/O system. Each table entry contains the address of a device driver. Position within the table corresponds to the logical device number.

Two device address tables exist, one for the 64R mode version of the run-time package (called DEVDAC), the other for 64V (DEVIP). A third file declares each device driver external name (DEVEXT). These tables reside in the UFD named FORMS*.

The contents of the 64R device address table is:

```
DAC      PR$IO      LDN 1 = PRINTER
OCT      0          NO ENTRY WITH LDN 2
DAC      V3$IO      LDN 3 = VISTAR3
DAC      OW$IO      LDN 4 = OWL1200
```

PRIME-SUPPLIED DEVICE DRIVERS

At present, FORMS supports the following three device drivers:

- offline printer (PRINTER)

- Infoton Vistar/3 (modified) (VISTAR3)
- Perkin-Elmer Owl (OWL1200)

Offline Printer Device Driver

The PRINTER device driver writes a form (or forms) to the line-printer spool queue. When the INVOKE command is issued to the line printer (IOCS Logical Unit 4), a file called PR##uu (where uu represents the user number) is opened. If it already exists, the file pointer is positioned to the end of file, where the new form definition will be written. If it does not exist, it is created after a record is written containing the control code for the line printer to enter FORTRAN forms-control mode.

When a form is output, one ASCII record is written for each line defined in the form. The first line contains a '1' in column 1, which causes the printer to eject to the top of a new page. Any enabled fields are underscored (with the _ character).

When the form is released, the file is copied into the spool queue, with the appropriate spool file header and file name. It is then closed and deleted from the home UFD. The PR##uu file must never appear in the home UFD after the program has not been completed; if it does, it means that the PRINTER form was not released.

Because of the new spool subsystem, two versions of PR$1IO are supplied. Source file PR$IO contains the version of the printer driver that is compatible with the spooler. This is the subroutine that is in the RFORMS and VFORMS libraries as released on the master disk. Source file OPR$IO contains the printer driver that is compatible with the older version spooler. To rebuild the FORMS libraries to work with the old spool subsystem, rename PR$IO to NPR$IO and then OPR$IO to PR$IO. The I/O system may then be rebuilt with C_IOR and C_IOV, after which the RFORMS and VFORMS libraries may be rebuilt with C_RLIB and C_VLIB, as outlined above.

VISTAR/3 Device Driver

The Infoton VISTAR/3 device driver (V3$IO) is designed around a specially modified VISTAR/3 (with microcode and hardware updates) available through Prime.

The device dimensions are 24 lines by 80 columns (1920 characters), all of which except the 15 character positions in the lower right of the screen are available for use by the application program. These character positions contain one of the following prompt or error messages from the device driver:

(spaces):
input not allowed

ENTER
enter data into unprotected fields on form.

press XMIT PAGE key when done

ERROR, RE-ENTER (blinking)
a character was lost on the last transmission -
press XMIT PAGE key

DATA ERROR (reverse video)
a field (or fields) does not meet the specified
validation criteria. The cursor is positioned to
the first character position of the erroneous field
correct the data and depress the 'XMIT PAGE' key

All unprotected fields are displayed surrounded by square brackets (i.e., [and]) and are displayed in full intensity. All protected fields are displayed at half intensity. Care must be taken to allow for the square brackets on unprotected fields when designing the form. The square brackets may be suppressed as an installation option by setting the variable ENCL in the device driver (FORMS>IOS>V3$IO) to zero.

To operate the VISTAR/3 with a program using FORMS, the switches in the rear of the display must be set as follows:

EOT character:	CR
mode:	block
line-speed	(user-selectable)
sec channel:	off
parity:	none
fdux/hdux:	(user-selectable)
stop bits:	2
roll/page:	roll

Owl Device Driver

The Perkin-Elmer OWL1200 device driver is designed for a stock OWL terminal which is capable of supporting function keys.

The device dimensions are 24 lines by 80 columns (1920) characters. The first character position, (1,1) and the last six character positions, (75,24) through (80,24), are not available for use by the form definition. Also, the character positions immediately preceding and following a field with any attribute other than protect must be vacant.

When an input operation occurs, the data on the screen may be transmitted to the computer by using any of the SEND keys on the right hand keypad. If function keys are disabled, striking F1 will also transmit the screen data. If function keys are enabled, striking any of the function keys will send the data to the computer and make available to the application program the number of the function key depressed. The number of function keys may be expanded two fold by using shift-Fn. This causes 16 to be added to the function key value.

When user input is required, one of the following prompt messages is printed in the lower right corner.

ENTER
Operator input is required - press one of the send or function keys when done.

DATA?
The data in the field to which the cursor is positioned does not conform to any of the validation criteria specified in the form definition. Re-enter the data and depress the SEND key.

SEQ?
The data was not transmitted from the terminal in the proper sequence. This usually indicates that a character was lost during transmission. Press the appropriate SEND or function key again.

SIZE?
Too many characters were sent for a given field in the form definition. This usually indicates that a character was lost during transmission. Press the appropriate SEND or function key again.

No special switch settings are required when a FORMS program is run on the OWL.

APPENDIX C

USER-WRITTEN DEVICE DRIVERS

INTRODUCTION

If the user desires to interface FORMS with a device or terminal that is not provided for by the standard interfaces described in Section 6, then it is necessary that the user write a device driver.

Techniques for this are discussed in the following paragraphs:

Terminal Requirements

Any terminal to be used with FORMS must have the following capabilities:

- internally buffered (block transmission) mode
- protected fields
- absolute cursor positioning
- data modification once displayed
- clear entire screen/clear unprotected data commands

Other features that could be taken advantage of by the FORMS system or device driver include:

- blink
- reverse video
- underlining
- keyboard lock
- input and/or output space compression

Device Driver Specification

Device drivers must be named xx$IO, where xx represents the two-character abbreviation used in the device control file. They have the following calling sequence:

    CALL xx$IO (function, iolist)

Function is one on the following nine function codes:

Function Number	Meaning
1	Initialize device: Reset all device logic, clear the entire screen, and enter block transmission mode (if this is a software function).
2	Output initial form: Write the contents of the entire I/O

data list (IOLIST) to the screen. The device driver should reset bits 1, 2, 3, and 4 of the attribute word for each entry and set bit 5 for each field displayed. It must not display any fields with the NODISPLAY bit 'bit 14) set. The location of the cursor following the output operation may be undefined.

3 Input form: First, the cursor must be positioned as follows:

- if DEVCM$ variable XPOS is zero, the cursor must be placed at the first character position of the first unprotected field displayed on the terminal. The DEVCM$ common block is described later.
- if XPOS is non-zero, the cursor must be positioned to location (XPOS, YPOS) on the device.

The device driver must wait for the user to fill in the displayed form and process the input as it is transmitted from the terminal. As it receives the data, the driver is responsible for inserting it into data areas in each field in the I/O list. Only fields with the DISPLAYED and ENABLED bits set in the attribute word must be input. On a fill duplex line, the device driver must disable the echo and auto-linefeed generation with a call to DUPLX$; this must be restored after the data has been input. If possible, a brief prompt message should be displayed in a convenient place on screen, informing the user that there is an input request pending.

If a function key was depressed, the device driver must check the logical variable FKEYS in the DEVCM$ common block. If OFF, refuse the function key request by waiting for the proper transmit key to be typed. If FKEYS is ON, save the function key number in the DEVCM$ variable FKEYNO (integer) and process the data as described above.

4 Modify existing form: The device driver must examine each entry in the I/O list and update those fields with attribute bits 1, 2, or 4 set. The recommended logic for the modify processor is:

- if data changed, enable/protect changed, or field attribute changed bits are all reset, process next field; else save current attribute word in a temporary storage area, and reset bits 1-4 (data/attributes modified) of the attribute word in IOLIST
- extract field length, and x,y, coordinates from IOLIST
- if the field is currently displayed and 'NODISPLAY' bit is set, erase this field from display, reset bit 5 in IOLIST entry, and process next field; else if field is

not currently displayed and 'NODISPLAY' bit is reset, display the field according to the supplied attributes and x,y, coordinates and set bit 5 in IOLIST entry; else if 'NODISPLAY' bit is set, ignore this field and process next; else if enable/protect changed bit is set and special handling is required to accomodate this change, perform this special handling.

- if attribute changed bit is set, update the field using the new attributes and process the next field; else update the data and process the next field.

5 Clear entire screen. All information displayed on the screen should be erased.

6 Clear unprotected data on screen. All unprotected information on the screen should be erased.

7 Close device: This function code is used to terminate device usage after a RELEASE command and is applicable primarily to the line-printer driver; terminal device drivers should switch the terminal back to conversational mode.

8 Correct data: The device driver must scan the I/O list for the first field with the 'data-invalid' attribute bit set (see below), position the cursor to the first character position of this field, and allow the operator to re-enter the data. It is recommended that an error/prompt message be displayed, informing the operator that the specified field has failed all validation tests and that it must be re-entered. This message should be displayed on a part of the screen that does not interfere with the form.

9 Print local: Write the contents of the entire screen to the local printer attached to the terminal; this feature must be supported by the particular terminal hardware in use. The device driver should return to the caller when the printer has completed printing.

Iolist is an array that contains the control and data definitions for each field in the form. It contains header words and at least 1 data word for each entry. The array must be accessed by the device driver using a pointer to the beginning of the field (supplied by the run-time package) added to an offset. This offset should be specified in the form of a PARAMETER'ed symbol, as defined below.

The following PARAMETERs represent each of the control words, plus the start of the data area. The device driver should be oblivious to their actual values, as these may change when new control information is added. The parameter declarations may be made thru a $INSERT file called 'IOPARM' in the directory containing the source of the I/O

system (e.g., as released, FORMS>IOS>IOPARM).

Parameter	Meaning
IOLK	Link to next entry in chain by position; this is not used by device drivers
IOVP	stream definition field pointer for this entry; this is not used by device drivers
IORP	format definition field pointer for this entry; this is not used by device drivers
IOSZ	field length, in characters
IOAT	field attributes, as follows:

Bit	Definition
1	Set by FORMS if data has changed since last display; reset be device driver when data has been updated on device.
2	Set by FORMS if enable/protect attribute has changed since last display; reset by device driver when field has been updated on device.
3	Set by FORMS if field has failed all supplied validation tests; reset by device driver when field has been re-entered from device.
4	Set by FORMS if any field attributes have been modified since last display; reset by device driver when field has been updated on device.
5	Set by device driver if field is currently displayed on device; reset by device driver if field is currently not displayed on device (initially reset).
13	Set by FORMS if field should be blinked when displayed; reset by FORMS if field should not be blinked when displayed.
14	Set by FORMS if field should not be displayed or should be erased if currently displayed; reset by FORMS if field should be displayed.
15	Set by FORMS if field should be displayed in reverse video; reset by FORMS if field should be displayed in normal video.
16	Set by FORMS if field should be write-enabled, (not protected); reset by FORMS if field should be write-protected.

IOYX line and column coordinates:
- left byte = line # (Y)
- right byte = column # (X)

IOPG physical page #; this is not used by device drivers

IODA start of text data; data is in ascii format, packed 2 characters per word, blank filled

The initialize, clear, close, and print functions (1, 5, 6, 7, and 9) are all relatively straightforward. These operations do not have to process data from the I/O list and therefore should assume it to be void.

The output, input, modify, and correct functions (2, 3, 4, and 8) all need to traverse the I/O list and process (or at least inspect) each field therein. The device driver must depend on the run-time system to provide a pointer for the start of each field definition in the I/O list. The run-time package contains two subroutines callable by the device driver for such a purpose. They are:

Subroutine	Function
FMS$RE	resets the internal (run-time package) field pointer to the beginning of the current page. This routine must be called at the beginning of the output, input, modify, and correct-data function processors. It may be called again to reset the pointer to the first field in the page when necessary (e.g, on an input error).
FMS$NF	returns the pointer to the next field in the I/O list to be processed. If the pointer is 0, the end of page or end of I/O list has been encountered. Fields are returned to the caller in line/column sequence.

FMS$RE Calling Sequence:

```
CALL FMS$RE
```

FMS$NF Calling Sequence:

```
CALL FMS$NF (pointer)
```

A common definition $INSERT file must be included in the device driver by the directive "$INSERT FORMS>RUN>DEVCM$". The common block contains 4 variables which are used by the input form (function 3) processor. They are:

Variable	Function
FKEYS	Logical variable set to true if function keys are enabled, false if disabled. If a function key is struck and FKEYS is false, the function key should be ignored; if true, the function key number should be stored in FKEYNO.
FKEYNO	16 bit integer which is set by the device driver to the number of the function key depressed. Should only be set if FKEYS is true.
XPOS	16 bit integer column number which the cursor is to be positioned prior to an input operation. If zero, position the cursor to the first enabled character position on the display.
YPOS	16 bit integer line number for cursor positioning prior to input. It is only valid of XPOS is non-zero.

A template for a device driver is included with the FORMS system. It is suggested that the user follow this template when writing a device driver.

INSTALLING THE DEVICE DRIVER

To install a new device driver into the FORMS run-time library, the user should follow the steps outlined below:

- Obtain a listing of the device control file and choose a free logical unit number above 10 (the first 10 are reserved by Prime). Append an entry to the DCF containing the selected logical unit number, device name, first 2 characters of the driver name (remember, the last 3 must be '$IO'), and the dimensions of the device in accordance with the format described in the section entitled "Device Mapping Scheme", above. For example, the Vistar/3 entry, whose logical unit number is 3, driver name is 'V3$IO', and dimensions are 24 by 80, would look as follows:

  3, VISTAR3, V3, 24, 80

- Attach to the directory containing the source for the Input/Output System and copy into it the source for the device driver to be installed.

- Edit the C_IOR (64R mode) and C_IOV (64V mode) command files and inset a line to compile the new device driver after the PR$IO routine.

- Run FAP and issue the GENERATE command to create the new device

tables and DCF which will include the new driver.

- Execute the C_IOR and/or C_IOV command file(s) to create a new input/output system.

- Attach to the first-level FORMS source directory ('FORMS') and execute the command file 'C_RLIB' to create a 64R mode library and/or C_VLIB to create a 64V mode library.

The user may now modify the TCB entries for the users that have the new terminal and reload application programs with the new version of the library. It is strongly recommended that the new library not be installed in the UFD named LIB until the new device driver has been tested.

APPENDIX D

TROUBLE SHOOTING

PROBLEM: FAP: IOLSIZ Overflow Error (F08)

SOLUTION: Change Default IOLSIZ To Necessary Size And Rebuild FAP.

1. In FORMS>FAP>IOBUF$:
   Change IOLSIZ Parameter (Default = 2500)
   To Necessary Size.

2. Rebuild FAP:
   Run Command Files In FORMS>FAP:
   - C_SUBS
   - C_FAP
   - C_LOAD

3. Copy *FAP Into CMDNC0 As FAP

4. Run FAP Again To Replace The FDL Binary

PROBLEM: Runtime I/O Error: IOLSIZ Overflow

SOLUTION: Change Default IOLSIZ To Necessary Size
And Rebuild FORMS Runtime Manager (RFORMS, VFORMS, SFORMS)

STEPS:

1. In FORMS>RUN>IOLDEF:
   Change IOLSIZ Parameter (Default = 2500)
   To Necessary Size.

*2. Rebuild Runtime Package:
   In FORMS>RUN, RUN Command File:
   - C_FMV
   - C_FMR

   In FORMS, Run Command File:
   - C_VLIB
   - C_RLIB

*3. Copy VFORMS, RFORMS Into LIB.

4. Reload Your Program.

*NOTE: If Running R-Mode Only Or
V-Mode Only,
Run Appropriate Command Files

PROBLEM: Adding A New Device Driver

SOLUTION: Add Driver Parameters To DCF.AS
And Rebuild IOS With New Driver.

STEPS:

1. In FORMS*:
   Edit DCF.AS
   Add Parameters For New Driver:
   For Example: To Add Logical Device Number 4,
   Device Name XY123, Device Driver Name XY, With
   24 Lines, 80 Colums; Insert The Line:
   4,XY123, XY, 24, 80

2. Run FAP Generate (Produces 3 $INSERT Files Which
   Contain Information About The New Driver Necessary
   In Building Runtime Library)

3. IN FORMS>IOS
   RUN C_IOR/C_IOV
   To Change BIOR/BIOV

4. In FORMS
   RUN C_VLIB, C_RLIB

5. Copy VFORMS/RFORMS To LIB

6. Execute Appropriate FAP TCB Command To Specify
   New Terminal For Specific User Numbers.
   (Refer To Section 4)

7. Reload Your Program With New Library.

APPENDIX E

SAMPLE FORTRAN PROGRAM

This appendix is a complete listing of the example program discussed in Section 7.

```
(0001)  C   ATSINP, FORMS, JRW, 78/02/23
(0002)  C   REVISED 1979 MLG, 79/06/06
(0003)  C   FORMS DEMO PROGRAM - INPUT ATS INFO, STORE IN DISK FILE
(0004)  C   COPYRIGHT 1979, PRIME COMPUTER INC, FRAMINGHAM
(0005)  C
(0006)  C
(0007)  C--- THIS PROGRAM INPUTS ATS INFORMATION FROM THE TERMINAL AND
(0008)  C    STORES THE INFO IN A DISK FILE.
(0009)  C
(0010)  C    TWO FILES ARE USED:
(0011)  C
(0012)  C       ATS.C   IS THE CONTROL FILE - IT CONTAINS THE NEXT ATS NUMBER TO BE
(0013)  C               ASSIGNED.
(0014)  C       ATS.D   IS THE DATA FILE.  AS EACH ATS FORM IS ENTERED, IT IS APPENDED
(0015)  C               TO THIS FILE IN THE FORMAT SHOWN IN THE PROGRAM.
(0016)  C
(0017)  C
(0018)  C--- TO TERMINATE THE PROGRAM, ENTER A NULL NAME FIELD.
(0019)  C
(0020)  C--- TO ENTER MORE THAN 4 ITEM LINES, ENTER A NON-SPACE CHARACTER (EXCEPT N)
(0021)  C    IN THE 'MORE' FIELD.
(0022)  C
(0023)  C--- THIS PROGRAM MAY BE USED BY A SINGLE USER IN ANY GIVEN DIRECTORY AT
(0024)  C    ONE TIME.  NO PROVISION IS MADE FOR CONCURRENT ACCESS TO THE DATA FILES.
(0025)  C
(0026)  C
(0027)  C
(0028)        COMMON /F$IOBF/ B(150)               /* EXTENDED I/O BUFFER
(0029)  C
(0030)        INTEGER NAMADR(75), VIA, HOW, REPL, INTC, BILL, SONUM(4),
(0031)       +        CHGNUM(4), CPO(4), ACOTHR(15), AIRSPR, INS(5),
(0032)       +        TYPE, CODE, NWIO, ATSNUM, B, I, J, MORE, FLD1(4,4),
(0033)       +        YESNOB(4), ACTBUF(4,3), FLD2(4,3)
(0034)  C
(0035)        INTEGER PART(8,4), DESCR(15,4), SN(4,4), QTY(4), RTN(4)
(0036)  C
(0037)        LOGICAL HDROUT
(0038)  C
(0039)  C
(0040)        EQUIVALENCE (YESNOB(1),REPL), (YESNOB(2),INTC),
(0041)       +            (YESNOB(3),BILL), (YESNOB(4),AIRSPR),
(0042)       +            (ACTBUF(1,1),SONUM), (ACTBUF(1,2),CHGNUM),
(0043)       +            (ACTBUF(1,3),CPO)
(0044)  C
(0045)  C
(0046)  C SYSCOM>KEYS.F      MNEMONIC KEYS FOR FILE SYSTEM (FTN)     31 MAY, 1977
(0046)       NOLIST
(0047)  C
(0048)  C
(0049)        DATA FLD1 /'REPL    ','INTC    ','BILL    ','AIRSPARE'/
(0050)        DATA FLD2 /'SONUM   ','CHGN    ','CPO     '/
(0051)  C
(0052)  C
```

```
(0053)  C---EXTEND TERMINAL, FILE I/O BUFFERS.
(0054)  C
(0055)        CALL ATTDEV(1,1,0,150)
(0056)        CALL ATTDEV(6,7,2,150)
(0057)  C
(0058)  C     CALL TO SHARED LIBRARY
(0059)        CALL FORM$I
(0060)  C
(0061)  C---OPEN FILES, INVOKE FORM ON TERMINAL.
(0062)  C
(0063)        CALL SRCH$$(K$CLOS,0,0,1,0,CODE)
(0064)        CALL SRCH$$(K$CLOS,0,0,2,0,CODE)
(0065)  C
(0066)        CALL SRCH$$(K$RDWR,'ATS.C',5,1,TYPE,CODE)
(0067)        IF (CODE.NE.0) CALL ERRPR$(K$NRTN,CODE,'ATS.C',5,0,0)
(0068)        CALL SRCH$$(K$RDWR,'ATS.D',5,2,TYPE,CODE)
(0069)        IF (CODE.NE.0) CALL ERRPR$(K$NRTN,CODE,'ATS.D',5,0,0)
(0070)  C
(0071)        CALL PRWF$$(K$POSN+K$PRER,2,LOC(0),0,10000000,NWIO,CODE)
(0072)  C
(0073)        WRITE (1,20)
(0074)  20    FORMAT('##INVOKE ADMN377')
(0075)  C
(0076)  C
(0077)  C---ASSIGN NEXT ATS #.
(0078)  C
(0079)  100   CALL PRWF$$(K$POSN+K$PREA,1,LOC(0),0,000000,NWIO,CODE)
(0080)        READ (5,120,ERR=160,END=160) ATSNUM
(0081)  120   FORMAT(I6)
(0082)        GO TO 200
(0083)  C
(0084)  C
(0085)  C---HERE ON EOF, ETC.
(0086)  C
(0087)  160   ATSNUM=0
(0088)  C
(0089)  C
(0090)  C---ASSIGN NEXT SEQUENTIAL ATS #.
(0091)  C
(0092)  180   ATSNUM=ATSNUM+1
(0093)  C
(0094)  C
(0095)  C---WRITE ATS#, CLEAR VARIABLE DATA, ERROR MESSAGE.
(0096)  C
(0097)  200   WRITE (1,210) ATSNUM
(0098)        HDROUT=.FALSE.                       /* HEADER NOT OUTPUT TO DISK FILE
(0099)  210   FORMAT('##SUBSTREAM HEADER'/I6/'##CLEAR'/'##SUBSTREAM ERROR'/' ')
(0100)  C
(0101)  C
(0102)  C---READ IN NAMES, ADDRESSES, AND ACCOUNTING INFORMATION.
(0103)  C
(0104)  220   READ (1,240) NAMADR, VIA, HOW, REPL, INTC, BILL, SONUM,
(0105)       +          CHGNUM, CPO, ACOTHR, AIRSPR, INS
```

```
(0106)  C
(0107)          IF(NAMADR(1).EQ.'  ') GO TO 5000     /* BLANK NAME => EXIT
(0108)  240     FORMAT(75A2/2I1,3A1,12A2,15A2,A1,4A2,A1)
(0109)  C
(0110)  C
(0111)  C---VALIDATE INPUT DATA.
(0112)  C
(0113)          IF (VIA.LT.1.OR.VIA.GT.9) GO TO 1000
(0114)          IF (HOW.LT.1.OR.HOW.GT.4) GO TO 1020
(0115)  C
(0116)  C
(0117)  C---CHECK YES/NO RESPONSES.
(0118)  C
(0119)          DO 250 I=1,4
(0120)          IF (YESNOB(I).GE.'a'.AND.YESNOB(I).LE.'z') /* MAP => UPPER CASE
(0121)        +    YESNOB(I)=AND(YESNOB(I),:157777)
(0122)          IF (YESNOB(I).NE.'Y'.AND.YESNOB(I).NE.'N') GO TO 1040
(0123)          IF (I.EQ.4) GO TO 250
(0124)          IF (YESNOB(I).EQ.'Y'.AND.ACTBUF(1,I).EQ.'  ') GO TO 1060
(0125)          IF (YESNOB(I).EQ.'N'.AND.ACTBUF(1,I).NE.'  ') GO TO 1080
(0126)  250     CONTINUE
(0127)  C
(0128)  C
(0129)  C---GET ITEM DATA.
(0130)  C
(0131)  400     READ (1,420) ((PART(J,I),J=1,8), (DESCR(J,I),J=1,15),
(0132)        +                (SN(J,I),J=1,4), QTY(I), RTN(I), I=1,4), MORE
(0133)  420     FORMAT(4(7A2,A1,15A2,4A2,I4,A1),A1)
(0134)  C
(0135)  C
(0136)  C---CHECK INPUT DATA VALIDITY.
(0137)  C
(0138)  500     DO 520 I=1,4
(0139)          IF (DESCR(1,I).EQ.'  ') GO TO 520    /* IGNORE BLANK LINE
(0140)          IF (RTN(I).GE.'a'.AND.RTN(I).LE.'z') /* MAP => UPPER CASE
(0141)        +    RTN(I)=AND(RTN(I),:157777)
(0142)          IF (RTN(I).NE.'Y'.AND.RTN(I).NE.'N') GO TO 1100
(0143)  520     CONTINUE
(0144)  C
(0145)  C
(0146)  C---WRITE DATA TO DISK FILE.
(0147)  C
(0148)          DO 550 I=1,4
(0149)          IF (DESCR(1,I).EQ.'  ') GO TO 550    /* IGNORE BLANK LINE
(0150)          IF (HDROUT) GO TO 540
(0151)  C
(0152)  C
(0153)  C---WRITE ACCOUNT HEADER TO DISK FILE.
(0154)  C
(0155)          WRITE (6,525) ATSNUM, NAMADR, VIA, HOW, REPL, INTC, BILL, SONUM,
(0156)        +               CHGNUM, CPO, ACOTHR, AIRSPR, INS
(0157)  525     FORMAT('*ATS',I6/5(15A2/),2I1,3A1,12A2/15A2/A1,4A2,A1)
(0158)  C
```

```
(0159)         HDROUT=.TRUE.
(0160)  C
(0161)  C
(0162)  C---WRITE INDIVIDUAL ITEM LINE TO DISK FILE.
(0163)  C
(0164)  C
(0165)  540    WRITE (6,545) (PART(J,I),J=1,8), (DESCR(J,I),J=1,15),
(0166)        +               (SN(J,I),J=1,4), QTY(I), RTN(I)
(0167)  545    FORMAT(7A2,A1,15A2,4A2,I4,A1)
(0168)  C
(0169)  550    CONTINUE
(0170)  C
(0171)  C
(0172)  C---CHECK FOR MORE ITEM LINES.
(0173)  C
(0174)         IF (MORE.EQ.' '.OR.MORE.EQ.'N'.OR.MORE.EQ.'n') GO TO 180
(0175)  C
(0176)         WRITE (1,580)
(0177)  580    FORMAT('##SUBSTREAM ITEMS'/' '/'##SUBSTREAM ITEMS'/
(0178)        +        '##POSITION PART01')
(0179)         GO TO 400                              /* NEXT SET OF ITEM LINES
(0180)  C
(0181)  C
(0182)  C---INCORRECT DATA IN ACCOUNTING FIELDS.
(0183)  C
(0184)  1000   WRITE (1,1010)
(0185)  1010   FORMAT('##SUBSTREAM ERROR'/
(0186)        +        'via code must be 1-9'/
(0187)        +        '##POSITION SHIPVIA')
(0188)         GO TO 220
(0189)  C
(0190)  1020   WRITE (1,1030)
(0191)  1030   FORMAT('##SUBSTREAM ERROR'/
(0192)        +        'how code must be 1-4'/
(0193)        +        '##POSITION SHIPHOW')
(0194)         GO TO 220
(0195)  C
(0196)  C
(0197)  C---YES/NO ANSWER REQ'D.
(0198)  C
(0199)  1040   WRITE (1,1050) (FLD1(J,I), J=1,4)
(0200)  1050   FORMAT('##SUBSTREAM ERROR'/
(0201)        +        'yes/no (Y or N) response required'/
(0202)        +        '##POSITION ',4A2)
(0203)         GO TO 220
(0204)  C
(0205)  C
(0206)  C---ACCUNT NUMBER FIELD BLANK.
(0207)  C
(0208)  1060   WRITE (1,1070) (FLD2(J,I), J=1,4)
(0209)  1070   FORMAT('##SUBSTREAM ERROR'/
(0210)        +        'account # required for YES response'/
(0211)        +        '##POSITION ',4A2)
```

```
(0212)         GO TO 220
(0213)  C
(0214)  C
(0215)  C---SURPLUS ACCOUNT NUMBER.
(0216)  C
(0217)  1080   WRITE (1,1090) (FLD2(J,I), J=1,4)
(0218)  1090   FORMAT('##SUBSTREAM ERROR'/
(0219)      +         'account # not permitted for NO response'/
(0220)      +         '##POSITION ',4A2)
(0221)         GO TO 220
(0222)  C
(0223)  C
(0224)  C---RETURN CODE FIELD BLANK.
(0225)  C
(0226)  1100   WRITE (1,1110) I
(0227)  1110   FORMAT('##SUBSTREAM ERROR'/
(0228)      +         'yes/no (Y or N) response required'/
(0229)      +         '##POSITION RTN',B'##'/
(0230)      +         '##SUBSTREAM ITEMS')
(0231)         GO TO 400
(0232)  C
(0233)  C
(0234)  C---HERE TO EXIT.  UPDATE ATS # IN CONTROL FILE.
(0235)  C
(0236)  5000   CALL PRWF$$(K$POSN+K$PREA,1,LOC(0),0,000000,NWIO,CODE)
(0237)         WRITE(5,120) ATSNUM
(0238)         CALL PRWF$$(K$TRNC,1,LOC(0),0,000000,NWIO,CODE)
(0239)  C
(0240)         CALL SRCH$$(K$CLOS,0,0,1,0,CODE)
(0241)         CALL SRCH$$(K$CLOS,0,0,2,0,CODE)
(0242)  C
(0243)         WRITE (1,5020)
(0244)  5020   FORMAT('##CLEAR ALL'/'##RELEASE')
(0245)  C
(0246)         CALL EXIT
(0247)  C
(0248)  C
(0249)  C
(0250)         END
PROGRAM SIZE:   PROCEDURE - 002564    LINKAGE - 000502    STACK - 000022
0000 ERRORS [<.MAIN.>FTN-REV15.3]
```

APPENDIX F

FORM DESCRIPTOR FOR FORTRAN PROGRAM EXAMPLE

This appendix is a complete listing of the example form descriptor associated with the program discussed in Section 7.

```
        PRIMEATS, FORMS, XXX, 79/02/12
(0001)  *       PRIMEATS, FORMS, XXX, 79/02/12
(0002)  *       PRIME AUTHORIZATION TO SHIP FORM — FORMS DEMO
(0003)  *       COPYRIGHT 1979, PRIME COMPUTER, FRAMINGHAM MA
(0004)  *
(0005)  *
(0006)  ADMN377  STREAM
(0007)  *
(0008)          LIST
(0009)  *
(0010)  *
(0011)  *--- HEADER INFORMATION.
(0012)  *
(0013)  HEADER  SUBSTREAM
(0014)          FIELD (FORMNAME,FORMNAME)
(0015)          FIELD ATSNUM, LENGTH 6, JUSTIFY RIGHT, ZERO-FILL, OUTPUT
(0016)          END SUBSTREAM
(0017)  *
(0018)  *
(0019)  *--- SHIP TO NAME AND ADDRESS.
(0020)  *
(0021)  NAMADR  SUBSTREAM
(0022)          FIELD NAME, LENGTH 30, VALIDATE 'P' OR 'B'
(0023)          REPEAT 3
(0024)          FIELD ADDR, LENGTH 30
(0025)          END REPEAT
(0026)  *
(0027)          FIELD ATTN, LENGTH 30
(0028)          END SUBSTREAM
(0029)  *
(0030)  *
(0031)  *--- SHIP VIA / HOW, ACCOUNTING, MISC INFO.
(0032)  *
(0033)  GENERAL  SUBSTREAM
(0034)          FIELD SHIPVIA, LENGTH 1, VALIDATE '9' OR 'B'
(0035)          FIELD SHIPHOW, LENGTH 1, VALIDATE '9' OR 'B'
(0036)  *
(0037)          FIELD REPL, LENGTH 1, VALIDATE 'A' OR 'B'
(0038)          FIELD INTC, LENGTH 1, VALIDATE 'A' OR 'B'
(0039)          FIELD BILL, LENGTH 1, VALIDATE 'A' OR 'B'
(0040)          FIELD SONUM, LENGTH 8, VALIDATE '99-99999' OR 'B'
(0041)          FIELD CHGN, LENGTH 8, VALIDATE '99-99999' OR 'B'
(0042)          FIELD CPO, LENGTH 8, VALIDATE '99-99999' OR 'B'
(0043)          FIELD ACCOTHER, LENGTH 30
(0044)  *
(0045)          FIELD AIRSPARE, LENGTH 1, VALIDATE 'A' OR 'B'
(0046)          FIELD INS, LENGTH 9, JUSTIFY RIGHT, ZERO-FILL, VALIDATE 'F'
(0047)          END SUBSTREAM
(0048)  *
(0049)  *
(0050)  *--- ITEM INFORMATION.
(0051)  *
(0052)  ITEMS  SUBSTREAM
```

```
(0053)          REPEAT 4
(0054)          FIELD PART, LENGTH 15, JUSTIFY RIGHT, SPACE-FILL
(0055)          FIELD DESCR, LENGTH 30, JUSTIFY LEFT
(0056)          FIELD SN, LENGTH 8, JUSTIFY RIGHT, ZERO-FILL
(0057)      FIELD QTY, LENGTH 4, JUSTIFY RIGHT, ZERO-FILL, VALIDATE '9' OR 'B'
(0058)          FIELD RTN, LENGTH 1, VALIDATE 'A' OR 'B'
(0059)          END REPEAT
(0060)          FIELD MORE, LENGTH 1, VALIDATE 'A' OR 'B'
(0061)          END SUBSTREAM
(0062)  *
(0063)  *
(0064)  *--- ERROR / WARNING MESSAGE.
(0065)  *
(0066)  ERROR  SUBSTREAM
(0067)          FIELD ERR, LENGTH 40, OUTPUT
(0068)          END SUBSTREAM
(0069)  *
(0070)  *
(0071)  *
(0072)          END STREAM
```

0000 ERRORS (FDL, REV 15 - 16-FEB-78)

I N P U T   S T R E A M   D E S C R I P T O R   STREAM: ADMN377

SUBSTREAM NAME	SUBSTREAM NUMBER	COLUMN BOUNDARIES	FIELD NAME	FIELD LENGTH
NAMADR	2	1- 30	NAME	30
NAMADR	2	31- 60	ADDR01	30
NAMADR	2	61- 90	ADDR02	30
NAMADR	2	91-120	ADDR03	30
NAMADR	2	121-150	ATTN	30
GENERAL	3	1	SHIPVIA	1
GENERAL	3	2	SHIPHOW	1
GENERAL	3	3	REPL	1
GENERAL	3	4	INTC	1
GENERAL	3	5	BILL	1
GENERAL	3	6- 13	SONUM	8
GENERAL	3	14- 21	CHGN	8
GENERAL	3	22- 29	CPO	8
GENERAL	3	30- 59	ACCOTHER	30
GENERAL	3	60	AIRSPARE	1
GENERAL	3	61- 69	INS	9
ITEMS	4	1- 15	PART01	15
ITEMS	4	16- 45	DESCR01	30
ITEMS	4	46- 53	SN01	8
ITEMS	4	54- 57	QTY01	4
ITEMS	4	58	RTN01	1
ITEMS	4	59- 73	PART02	15
ITEMS	4	74-103	DESCR02	30
ITEMS	4	104-111	SN02	8
ITEMS	4	112-115	QTY02	4

ITEMS	4	116	RTN02	1
ITEMS	4	117-131	PART03	15
ITEMS	4	132-161	DESCR03	30
ITEMS	4	162-169	SN03	8
ITEMS	4	170-173	QTY03	4
ITEMS	4	174	RTN03	1
ITEMS	4	175-189	PART04	15
ITEMS	4	190-219	DESCR04	30
ITEMS	4	220-227	SN04	8
ITEMS	4	228-231	QTY04	4
ITEMS	4	232	RTN04	1
ITEMS	4	233	MORE	1

O U T P U T   S T R E A M   D E S C R I P T O R   STREAM: ADMN377

SUBSTREAM NAME	SUBSTREAM NUMBER	COLUMN BOUNDARIES	FIELD NAME	FIELD LENGTH
HEADER	1	1- 6	ATSNUM	6
NAMADR	2	1- 30	NAME	30
NAMADR	2	31- 60	ADDR01	30
NAMADR	2	61- 90	ADDR02	30
NAMADR	2	91-120	ADDR03	30
NAMADR	2	121-150	ATTN	30
GENERAL	3	1	SHIPVIA	1
GENERAL	3	2	SHIPHOW	1
GENERAL	3	3	REPL	1
GENERAL	3	4	INTC	1
GENERAL	3	5	BILL	1
GENERAL	3	6- 13	SONUM	8
GENERAL	3	14- 21	CHGN	8
GENERAL	3	22- 29	CPO	8
GENERAL	3	30- 59	ACCOTHER	30
GENERAL	3	60	AIRSPARE	1
GENERAL	3	61- 69	INS	9
ITEMS	4	1- 15	PART01	15
ITEMS	4	16- 45	DESCR01	30
ITEMS	4	46- 53	SN01	8
ITEMS	4	54- 57	QTY01	4
ITEMS	4	58	RTN01	1
ITEMS	4	59- 73	PART02	15
ITEMS	4	74-103	DESCR02	30
ITEMS	4	104-111	SN02	8
ITEMS	4	112-115	QTY02	4
ITEMS	4	116	RTN02	1
ITEMS	4	117-131	PART03	15
ITEMS	4	132-161	DESCR03	30
ITEMS	4	162-169	SN03	8
ITEMS	4	170-173	QTY03	4
ITEMS	4	174	RTN03	1
ITEMS	4	175-189	PART04	15
ITEMS	4	190-219	DESCR04	30
ITEMS	4	220-227	SN04	8
ITEMS	4	228-231	QTY04	4
ITEMS	4	232	RTN04	1
ITEMS	4	233	MORE	1
ERROR	5	1- 40	ERR	40

```
      PRIMEATS, FORMS XXX, 79/02/12
(0073)   *    PRIMEATS, FORMS XXX, 79/02/12
(0074)   *    PRIME AUTHORIZATION TO SHIP FORM -- FORMS DEMO
(0075)   *    COPYRIGHT 1979 PRIME COMPUTER, FRAMINGHAM MA
(0076)   *
(0077)   *
(0078)   ADMN377      FORMAT
(0079)            DEVICE OWL1200
(0080)   *
(0081)   *--- HEADER LINE INFORMATION:
(0082)   *
(0083)              FIELD 'FORM' POSITION (2,1)
(0084)   FORMNAME            FIELD LENGTH 8, POSITION (7,1)
(0085)            FIELD 'ATS #' POSITION (20,1)
(0086)   ATSNUM          FIELD LENGTH 6, POSITION (26,1)
(0087)   *
(0088)   *
(0089)   *--- SHIP TO INFORMATION:
(0090)   *
(0091)              FIELD 'SHIP TO ' POSITION (2,3), REVERSE VIDEO
(0092)              FIELD 'NAME' POSITION (12,3)
(0093)   NAME          FIELD LENGTH 30, POSITION (24,3), NOPROTECT
(0094)              FIELD 'ADDRESS' POSITION (12,4)
(0095)              REPEAT 3
(0096)   ADDR          FIELD LENGTH 30, POSITION (24,+3), NOPROTECT
(0097)              END REPEAT
(0098)              FIELD 'ATTENTION' POSITION (12,7)
(0099)   ATTN          FIELD LENGTH 30 POSITION (24,7) NOPROTECT
(0100)   *
(0101)   *
(0102)   *--- SHIP VIA INFORMATION:
(0103)   *
(0104)              FIELD 'SHIP VIA' POSITION (2,9), REVERSE VIDEO
(0105)   SHIPVIA           FIELD LENGTH 1, POSITION (12,9), NOPROTECT
(0106)              FIELD '**VIA CODES**' POSITION (62,1)
(0107)              FIELD '1. PICKUP' POSITION (62,2)
(0108)              FIELD '2. PARCEL POSITIONT' POSITION (62,3)
(0109)              FIELD  '3. UPS' POSITION (62,4)
(0110)              FIELD '4. FIRST CLASS' POSITION (62,5)
(0111)              FIELD '5. SPEC DELIV' POSITION (62,6)
(0112)              FIELD '6. TRUCK' POSITION (62,7)
(0113)              FIELD '7. PRI PARCEL' POSITION (62,8)
(0114)              FIELD '8. AIR FREIGHT' POSITION (62,9)
(0115)              FIELD '9. FEDR EXPR' POSITION (62,10)
(0116)   *
(0117)   *
(0118)   *---SHIP HOW INFORMATION:
(0119)   *
(0120)              FIELD 'SHIP HOW' POSITION (20,9), REVERSE VIDEO
(0121)   SHIPHOW           FIELD LENGTH 1, POSITION (32,9), NOPROTECT
(0122)              FIELD '**HOW CODES**' POSITION (62,12)
(0123)              FIELD '1. PREPAID', POSITION (62,13)
(0124)              FIELD '2. C.O.D.', POSITION (62,14)
```

```
(0125)              FIELD '3. PREPAID/ADD', POSITION (62,15)
(0126)              FIELD '4. COLLECT', POSITION (62,16)
(0127)    *
(0128)    *
(0129)    *--- ACCOUNTING INFORMATION:
(0130)    *
(0131)              FIELD 'ACCOUNT ' POSITION (2,11), REVERSE VIDEO
(0132)              FIELD 'REPLACE/SHORT SHIP?' POSITION (12,11)
(0133)    REPL         FIELD LENGTH 1, POSITION (33,11), NOPROTECT
(0134)              FIELD 'S.O.', POSITION (37,11)
(0135)    SONUM         FIELD LENGTH 8, POSITION (50,11), NOPROTECT
(0136)              FIELD 'INTERNAL CHARGE?' POSITION (12,12)
(0137)    INTC         FIELD LENGTH 1, POSITION (33,12), NOPROTECT
(0138)              FIELD 'CHARGE #' POSITION (37,12)
(0139)    CHGN         FIELD LENGTH 8, POSITION (50,12), NOPROTECT
(0140)              FIELD 'BILLABLE?', POSITION (12,13)
(0141)    BILL         FIELD LENGTH 1, POSITION (33,13), NOPROTECT
(0142)              FIELD  'COST P.O. #', POSITION (37,13)
(0143)    CPO         FIELD LENGTH 8, POSITION (50,13), NOPROTECT
(0144)              FIELD  'OTHER:' POSITION (12,14)
(0145)    ACCOTHER          FIELD LENGTH 30, POSITION (20,14) NOPROTECT
(0146)    *
(0147)    *
(0148)    *--- MISCELLANEOUS INFORMATION:
(0149)    *
(0150)              FIELD 'MISC     ' POSITION (2,16), REVERSE VIDEO
(0151)              FIELD 'INSURE FOR   $' POSITION (12,16)
(0152)    INS         FIELD LENGTH 9, POSITION (28,16),NOPROTECT
(0153)              FIELD 'AIR SPARE?' POSITION (40,16)
(0154)    AIRSPACE          FIELD LENGTH 1, POSITION (52,16), NOPROTECT
(0155)    *
(0156)    *
(0157)    *--- ITEM DESCRIPTION.
(0158)    *
(0159)              FIELD 'PART NO' POSITION (5,18)
(0160)              FIELD 'DESCRIPTION' POSITION (29,18)
(0161)              FIELD 'S/N' POSITION (53,18)
(0162)              FIELD 'QTY' POSITION (61,18)
(0163)              FIELD 'RTN' POSITION (66,18)
(0164)              REPEAT 4
(0165)    PART         FIELD LENGTH 15, POSITION (2,+18), NOPROTECT
(0166)    DESCR         FIELD LENGTH 30, POSITION (19,+18), NOPROTECT
(0167)    SN          FIELD LENGTH 8, POSITION (51,+18), NOPROTECT
(0168)    QTY          FIELD LENGTH 4, POSITION (61,+18), NOPROTECT
(0169)    RTN          FIELD LENGTH 1, POSITION (67,+18), NOPROTECT
(0170)              END REPEAT
(0171)    *
(0172)              FIELD 'MORE?' POSITION (72,22)
(0173)    MORE         FIELD LENGTH 1, POSITION (79,22), NOPROTECT
(0174)    *
(0175)    *
(0176)    *--- ERROR MESSAGE, ETC.
(0177)    *
```

```
(0178)  ERR          FIELD LENGTH 40, POSITION (2,24)
(0179)  *
(0180)  *
(0181)  *
(0182)              END DEVICE
(0183)            DEVICE VISTAR3
(0184)  *
(0185)  *--- HEADER LINE INFORMATION:
(0186)  *
(0187)              FIELD 'FORM' POSITION (2,1)
(0188)  FORMNAME           FIELD LENGTH 8, POSITION (7,1)
(0189)            FIELD 'ATS #' POSITION (20,1)
(0190)  ATSNUM           FIELD LENGTH 6, POSITION (26,1)
(0191)  *
(0192)  *
(0193)  *--- SHIP TO INFORMATION:
(0194)  *
(0195)              FIELD 'SHIP TO ' POSITION (2,3), REVERSE VIDEO
(0196)              FIELD 'NAME' POSITION (12,3)
(0197)  NAME           FIELD LENGTH 30, POSITION (24,3), NOPROTECT
(0198)              FIELD 'ADDRESS' POSITION (12,4)
(0199)              REPEAT 3
(0200)  ADDR           FIELD LENGTH 30, POSITION (24,+3), NOPROTECT
(0201)              END REPEAT
(0202)              FIELD 'ATTENTION' POSITION (12,7)
(0203)  ATTN           FIELD LENGTH 30 POSITION (24,7) NOPROTECT
(0204)  *
(0205)  *
(0206)  *--- SHIP VIA INFORMATION:
(0207)  *
(0208)              FIELD 'SHIP VIA' POSITION (2,9), REVERSE VIDEO
(0209)  SHIPVIA           FIELD LENGTH 1, POSITION (12,9), NOPROTECT
(0210)              FIELD '**VIA CODES**' POSITION (62,1)
(0211)              FIELD '1. PICKUP' POSITION (62,2)
(0212)              FIELD '2. PARCEL POSITIONT' POSITION (62,3)
(0213)              FIELD  '3. UPS' POSITION (62,4)
(0214)              FIELD '4. FIRST CLASS' POSITION (62,5)
(0215)              FIELD '5. SPEC DELIV' POSITION (62,6)
(0216)              FIELD '6. TRUCK' POSITION (62,7)
(0217)              FIELD '7. PRI PARCEL' POSITION (62,8)
(0218)              FIELD '8. AIR FREIGHT' POSITION (62,9)
(0219)              FIELD '9. FEDR EXPR' POSITION (62,10)
(0220)  *
(0221)  *
(0222)  *---SHIP HOW INFORMATION:
(0223)  *
(0224)              FIELD 'SHIP HOW' POSITION (20,9), REVERSE VIDEO
(0225)  SHIPHOW           FIELD LENGTH 1, POSITION (32,9), NOPROTECT
(0226)              FIELD '**HOW CODES**' POSITION (62,12)
(0227)              FIELD '1. PREPAID', POSITION (62,13)
(0228)              FIELD '2. C.O.D.', POSITION (62,14)
(0229)              FIELD '3. PREPAID/ADD', POSITION (62,15)
(0230)              FIELD '4. COLLECT', POSITION (62,16)
```

```
(Ø231)  *
(Ø232)  *
(Ø233)  *--- ACCOUNTING INFORMATION:
(Ø234)  *
(Ø235)            FIELD 'ACCOUNT ' POSITION (2,11), REVERSE VIDEO
(Ø236)            FIELD 'REPLACE/SHORT SHIP?' POSITION (12,11)
(Ø237)  REPL         FIELD LENGTH 1, POSITION (33,11), NOPROTECT
(Ø238)            FIELD 'S.O.', POSITION (37,11)
(Ø239)  SONUM         FIELD LENGTH 8, POSITION (5Ø,11), NOPROTECT
(Ø240)            FIELD 'INTERNAL CHARGE?' POSITION (12,12)
(Ø241)  INTC         FIELD LENGTH 1, POSITION (33,12), NOPROTECT
(Ø242)            FIELD 'CHARGE #' POSITION (37,12)
(Ø243)  CHGN         FIELD LENGTH 8, POSITION (5Ø,12), NOPROTECT
(Ø244)            FIELD 'BILLABLE?', POSITION (12,13)
(Ø245)  BILL         FIELD LENGTH 1, POSITION (33,13), NOPROTECT
(Ø246)            FIELD  'COST P.O. #', POSITION (37,13)
(Ø247)  CPO         FIELD LENGTH 8, POSITION (5Ø,13), NOPROTECT
(Ø248)            FIELD  'OTHER:' POSITION (12,14)
(Ø249)  ACCOTHER          FIELD LENGTH 3Ø, POSITION (2Ø,14) NOPROTECT
(Ø25Ø)  *
(Ø251)  *
(Ø252)  *--- MISCELLANEOUS INFORMATION:
(Ø253)  *
(Ø254)            FIELD 'MISC    ' POSITION (2,16), REVERSE VIDEO
(Ø255)            FIELD 'INSURE FOR   $' POSITION (12,16)
(Ø256)  INS         FIELD LENGTH 9, POSITION (28,16),NOPROTECT
(Ø257)            FIELD 'AIR SPARE?' POSITION (4Ø,16)
(Ø258)  AIRSPACE          FIELD LENGTH 1, POSITION (52,16), NOPROTECT
(Ø259)  *
(Ø26Ø)  *
(Ø261)  *--- ITEM DESCRIPTION.
(Ø262)  *
(Ø263)            FIELD 'PART NO' POSITION (5,18)
(Ø264)            FIELD 'DESCRIPTION' POSITION (29,18)
(Ø265)            FIELD 'S/N' POSITION (53,18)
(Ø266)            FIELD 'QTY' POSITION (61,18)
(Ø267)            FIELD 'RTN' POSITION (66,18)
(Ø268)            REPEAT 4
(Ø269)  PART         FIELD LENGTH 15, POSITION (2,+18), NOPROTECT
(Ø27Ø)  DESCR         FIELD LENGTH 3Ø, POSITION (19,+18), NOPROTECT
(Ø271)  SN          FIELD LENGTH 8, POSITION (51,+18), NOPROTECT
(Ø272)  QTY         FIELD LENGTH 4, POSITION (61,+18), NOPROTECT
(Ø273)  RTN         FIELD LENGTH 1, POSITION (67,+18), NOPROTECT
(Ø274)            END REPEAT
(Ø275)  *
(Ø276)            FIELD 'MORE?' POSITION (72,22)
(Ø277)  MORE         FIELD LENGTH 1, POSITION (79,22), NOPROTECT
(Ø278)  *
(Ø279)  *
(Ø28Ø)  *--- ERROR MESSAGE, ETC.
(Ø281)  *
(Ø282)  ERR         FIELD LENGTH 4Ø, POSITION (2,24)
(Ø283)  *
```

```
(Ø284)   *
(Ø285)   *
(Ø286)              END DEVICE
(Ø287)              END FORMAT

ØØØØ ERRORS (FDL, REV 15 - 16-FEB-78)
```

```
 D E V I C E   F O R M A T   M A P     FORMAT: ADMN377    DEVICE: CWL1200    SIZE: 24 BY 80    PAGE: 1
Ø
Ø
       ....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*....8
     ----------------------------------------------------------------------------------
Ø  1 | FORM ********     ATS # ******                            **VIA CODES**       |
Ø  2 |                                                           1. PICKUP           |
Ø  3 | SHIP TO   NAME                                            2. PARCEL POSITIONT |
+                       ______________________________
Ø  4 |           ADDRESS                                         3. UPS              |
+                       ______________________________
Ø  5 |                                                           4. FIRST CLASS      |
+                       ______________________________
Ø  6 |                                                           5. SPEC DELIV       |
+                       ______________________________
Ø  7 |           ATTENTION                                       6. TRUCK            |
+                       ______________________________
Ø  8 |                                                           7. PRI PARCEL       |
Ø  9 | SHIP VIA          SHIP HOW                                8. AIR FREIGHT      |
+               _                   _
Ø 10 |                                                           9. FEDR EXPR        |
Ø 11 | ACCOUNT   REPLACE/SHORT SHIP?      S.O.                                       |
+                                _                  ________
Ø 12 |           INTERNAL CHARGE?         CHARGE #               **HOW CODES**       |
+                                _                  ________
Ø 13 |           BILLABLE?                COST P.O. #            1. PREPAID          |
+                                _                  ________
Ø 14 |           OTHER:                                          2. C.O.D.           |
+                   ______________________________
Ø 15 |                                                           3. PREPAID/ADD      |
Ø 16 | MISC      INSURE FOR   $              AIR SPARE?           4. COLLECT          |
+                           _________             _
Ø 17 |                                                                               |
Ø 18 |    PART NO                DESCRIPTION             S/N      QTY  RTN           |
Ø 19 |                                                                               |
+      _______________ ______________________________ ________ ____ _
Ø 20 |                                                                               |
+      _______________ ______________________________ ________ ____ _
Ø 21 |                                                                               |
+      _______________ ______________________________ ________ ____ _
Ø 22 |                                                                     MORE?     |
+      _______________ ______________________________ ________ ____ _           _
Ø 23 |                                                                               |
Ø 24 | ****************************************                                      |
Ø    ----------------------------------------------------------------------------------
```

```
 D E V I C E   F O R M A T   M A P     FORMAT: ADMN377    DEVICE: VISTAR3     SIZE: 24 BY 80     PAGE: 1
0
0
       ....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*....8
     --------------------------------------------------------------------------------
0  1 | FORM ********      ATS # ******                             **VIA CODES**        |
0  2 |                                                              1. PICKUP            |
0  3 | SHIP TO  NAME                                                2. PARCEL POSITIONT  |
+                      ____________________________
0  4 |          ADDRESS                                             3. UPS               |
+                      ____________________________
0  5 |                                                              4. FIRST CLASS       |
+                      ____________________________
0  6 |                                                              5. SPEC DELIV        |
+                      ____________________________
0  7 |          ATTENTION                                           6. TRUCK             |
+                      ____________________________
0  8 |                                                              7. PRI PARCEL        |
0  9 | SHIP VIA          SHIP HOW                                   8. AIR FREIGHT       |
+               _                 _
0 10 |                                                              9. FEDR EXPR         |
0 11 | ACCOUNT  REPLACE/SHORT SHIP?      S.O.                                            |
+                                _                  ________
0 12 |          INTERNAL CHARGE?         CHARGE #                   **HOW CODES**        |
+                                _                  ________
0 13 |          BILLABLE?                COST P.O. #                1. PREPAID           |
+                                _                  ________
0 14 |          OTHER:                                              2. C.O.D.            |
+                     ___________________________
0 15 |                                                              3. PREPAID/ADD       |
0 16 | MISC     INSURE FOR    $                AIR SPARE?           4. COLLECT           |
+                             ________                _
0 17 |                                                                                   |
0 18 |    PART NO                DESCRIPTION              S/N     QTY  RTN                |
0 19 |                                                                                   |
+      ______________  ____________________________  _______  ____  _
0 20 |                                                                                   |
+      ______________  ____________________________  _______  ____  _
0 21 |                                                                                   |
+      ______________  ____________________________  _______  ____  _
0 22 |                                                                    MORE?          |
+      ______________  ____________________________  _______  ____  _            _
0 23 |                                                                                   |
0 24 | ****************************************                                          |
0    --------------------------------------------------------------------------------
```

APPENDIX G

SAMPLE COBOL PROGRAM LISTING

This appendix is a complete listing of the example program discussed in Section 8.

```
REV 15.3  COBOL     SOURCE FILE:  DEMO1                              08/13/79  13:52
(0001)          IDENTIFICATION DIVISION.
(0002)          PROGRAM-ID. MAIN.
(0003)          REMARKS.  A PROGRAM TO ACCEPT ORDERS FROM THE VDU
(0004)                    AND WRITE OUT A MIDAS FILE.
(0005)          ENVIRONMENT DIVISION.
(0006)          INPUT-OUTPUT SECTION.
(0007)          FILE-CONTROL.
(0008)              SELECT INFILE ASSIGN TO TERMINAL.
(0009)              SELECT ORDER-FILE ASSIGN TO PFMS
(0010)                ORGANIZATION IS INDEXED
(0011)                ACCESS IS DYNAMIC
(0012)                RECORD KEY IS ORDER-KEY
(0013)                ALTERNATE RECORD KEY IS ORDER-NAME WITH DUPLICATES
(0014)                ALTERNATE RECORD KEY IS STOCK-NO WITH DUPLICATES
(0015)                ALTERNATE RECORD KEY IS STOCK-DEL WITH DUPLICATES.
(0016)          DATA DIVISION.
(0017)          FILE SECTION.
(0018)          FD  INFILE  LABEL RECORDS ARE OMITTED.
(0019)          01  SCR.
(0020)              02  FILLER         PIC X(104).
(0021)          FD ORDER-FILE LABEL RECORDS ARE STANDARD
(0022)                    VALUE OF FILE-ID IS "ORDERS".
(0023)          01 ORDER-RECORD.
(0024)              02 ORDER-KEY.
(0025)                  03  ORDER-NO           PIC 9(5).
(0026)                  03  ORDER-ITEM         PIC 99.
(0027)              02 ORDER-NAME      PIC X(20).
(0028)              02 ORDER-ADD1      PIC X(20).
(0029)              02 ORDER-ADD2      PIC X(20).
(0030)              02 ORDER-ADD3      PIC X(20).
(0031)              02 STOCK-NO        PIC X(6).
(0032)              02 STOCK-DEL       PIC X(8).
(0033)              02 STOCK-QTY       PIC S9(5) SIGN TRAILING SEPARATE.
(0034)         01  ORDERC.
(0035)             02 FILLER           PIC X(7).
(0036)             02 NEXT-ORDER       PIC 9(5).
(0037)             02 FILLER           PIC X(94).
(0038)          WORKING-STORAGE SECTION.
(0039)         77  INV-C  PIC X(12)  VALUE '##INVOKE DS1'.
(0040)         77  REL-C    PIC X(09)  VALUE '##RELEASE'
(0041)         77  PROT-C   PIC X(25)  VALUE '##PROTECT NAM AD1 AD2 AD3'.
(0042)         77  ENAB-C   PIC X(24)  VALUE '##ENABLE NAM AD1 AD2 AD3'.
(0043)         77  CLEAR-C  PIC X(07) VALUE '##CLEAR'.
(0044)         77  CLA-C    PIC X(11) VALUE '##CLEAR ALL'.
(0045)         77  CURR-ORD     PIC 9(5).
(0046)         01  DATA-SCREEN.
(0047)             02  DS-NAM.
(0048)                 03  CUSIND    PIC XX.
(0049)                 03  FILLER    PIC X(18).
(0050)             02  DS-AD1      PIC X(20).
(0051)             02  DS-AD2      PIC X(20).
(0052)             02  DS-AD3      PIC X(20).
```

```
(0053)              02  DS-STK.
(0054)                  03  STK-ID      PIC XX.
(0055)                  03  FILLER      PIC X(4).
(0056)              02  DS-QTY        PIC 9(5).
(0057)              02  DS-DATE       PIC X(8).
(0058)              02  DS-ORDER      PIC 9(5).
(0059)           PROCEDURE DIVISION.
(0060)           START-POINT.
(0061)               CALL 'FORM$I'.
(0062)               OPEN I-O ORDER-FILE.
(0063)               OPEN I-O INFILE.
(0064)               WRITE SCR FROM INV-C.
(0065)           B-POINT.
(0066)              READ INFILE INTO DATA-SCREEN.
(0067)              IF CUSIND EQUAL '**' GO TO END-IT.
(0068)        *
(0069)        *
(0070)          TRY-IT.
(0071)              MOVE ZEROES TO ORDER-KEY.
(0072)              READ ORDER-FILE KEY IS ORDER-KEY INVALID KEY
(0073)                   GO TO TRY-IT.
(0074)              ADD 1 TO NEXT-ORDER.
(0075)              MOVE NEXT-ORDER TO CURR-ORD.
(0076)              REWRITE ORDER-RECORD.
(0077)        *
(0078)        *
(0079)              MOVE DS-NAM TO ORDER-NAME.
(0080)              MOVE DS-AD1 TO ORDER-ADD1.
(0081)              MOVE DS-AD2 TO ORDER-ADD2.
(0082)              MOVE DS-AD3 TO ORDER-ADD3.
(0083)              MOVE ZERO TO ORDER-ITEM.
(0084)          A-POINT.
(0085)              MOVE DS-STK TO STOCK-NO.
(0086)              INSPECT DS-STK REPLACING LEADING SPACES BY ZEROES.
(0087)              MOVE DS-QTY TO STOCK-QTY.
(0088)              MOVE DS-DATE TO STOCK-DEL.
(0089)        *
(0090)        *
(0091)              ADD 1 TO ORDER-ITEM.
(0092)              MOVE CURR-ORD TO ORDER-NO.
(0093)              MOVE CURR-ORD TO DS-ORDER.
(0094)        *
(0095)        *
(0096)              WRITE ORDER-RECORD.
(0097)              WRITE SCR FROM PROT-C.
(0098)              MOVE SPACES TO DS-STK, DS-QTY, DS-DATE.
(0099)              WRITE SCR FROM DATA-SCREEN.
(0100)              READ INFILE INTO DATA-SCREEN.
(0101)              IF STK-ID NOT EQUAL '**' GO TO A-POINT.
(0102)        *
(0103)        *
(0104)              WRITE SCR FROM ENAB-C.
(0105)              WRITE SCR FROM CLEAR-C.
```

```
(0106)          GO TO B-POINT.
(0107)      *
(0108)      *
(0109)       END-IT.
(0110)          CLOSE ORDER-FILE.
(0111)          WRITE SCR FROM CLA-C.
(0112)          WRITE SCR FROM REL-C.
(0113)          CLOSE INFILE.
(0114)          STOP 'END OF ORDER ENTRY'.
```

```
0071  /W/ MOVE IS DONE WITHOUT CONVERSION.
```

```
P R O G R A M   S T A T I S T I C S

EXECUTABLE CODE SIZE: 453 WORDS.
CONSTANT POOL SIZE: 35 WORDS.
TOTAL PURE PROCEDURE SIZE: 488 WORDS.

WORKING-STORAGE SIZE: 202 BYTES.
TOTAL LINKFRAME SIZE: 499 WORDS.

STACK SIZE: 29 WORDS.

TRACE MODE: OFF.

NO ARGUMENTS EXPECTED.

114 SOURCE LINES.


NO ERRORS,  1 WARNINGS,  P400/500 COBOL REV 15.3  <MAIN  >
```

APPENDIX H

FORM DESCRIPTOR FOR COBOL PROGRAM EXAMPLE

This appendix is a complete listing of the example form descriptor associated with the program discussed in Section 8.

```
(0001)  DS1    STREAM
(0002)  NAM    FIELD NAM LENGTH 20
(0003)  AD1    FIELD AD1 LENGTH 20
(0004)  AD2    FIELD AD2 LENGTH 20
(0005)  AD3    FIELD AD3 LENGTH 20
(0006)  STKNNO FIELD STKNO LENGTH 6
(0007)  STKQTY FIELD STKQTY LENGTH 5 JUSTIFY RIGHT
(0008)  STKDEL FIELD STKDEL LENGTH 8
(0009)  ONO    FIELD ONO LENGTH 5 OUTPUT
(0010)         END STREAM

0000 ERRORS (FDL, REV 16 - 16-FEB-79)
```

I N P U T   S T R E A M   D E S C R I P T O R   STREAM: DS1

COLUMN BOUNDARIES	FIELD NAME	FIELD LENGTH
1- 20	NAM	20
21- 40	AD1	20
41- 60	AD2	20
61- 80	AD3	20
81- 86	STKNO	6
87- 91	STKQTY	5
92- 99	STKDEL	8

OUTPUT STREAM DESCRIPTOR STREAM: DS1

COLUMN BOUNDARIES	FIELD NAME	FIELD LENGTH
1- 20	NAM	20
21- 40	AD1	20
41- 60	AD2	20
61- 80	AD3	20
81- 86	STKNO	6
87- 91	STKQTY	5
92- 99	STKDEL	8
100-104	ONO	5

```
(0011)    ***********
(0012)    ***********
(0013)    ***********
(0014)    DS1    FORMAT
(0015)           DEVICE VISTAR3
(0016)           FIELD 'O R D E R   E N T R Y   S Y S T E M' POSITION (22,2)
(0017)           FIELD '- - - - -   - - - - -   - - - - - - -' POSITION (22,3)
(0018)           FIELD 'ACCOUNT NAME    ' POSITION (10,6)
(0019)    NAM    FIELD LENGTH 20 POSITION (35,6) NOPROTECT
(0020)           FIELD '    AND ADDRESS' POSITION (10,7)
(0021)    AD1    FIELD LENGTH 20 POSITION (35,7) NOPROTECT
(0022)    AD2    FIELD LENGTH 20 POSITION (35,8) NOPROTECT
(0023)    AD3    FIELD LENGTH 20 POSITION (35,9) NOPROTECT
(0024)           FIELD 'STOCK NUMBER' POSITION (10,12)
(0025)    STKNO  FIELD LENGTH 6 POSITION (35,12) NOPROTECT
(0026)           FIELD 'QUANTITY' POSITION (10,14)
(0027)    STKQTY FIELD LENGTH 5 POSITION (35,14) NOPROTECT
(0028)           FIELD 'DELIVERY DATE' POSITION (10,16)
(0029)    STKDEL FIELD LENGTH 8 POSITION (35,16) NOPROTECT
(0030)           FIELD 'ORDER REFERENCE IS' POSITION (10,20)
(0031)    ONO    FIELD LENGTH 5 POSITION (35,20)
(0032)           END DEVICE
(0033)           DEVICE OWL1200
(0034)           FIELD 'O R D E R   E N T R Y   S Y S T E M' POSITION (22,2)
(0035)           FIELD '- - - - -   - - - - -   - - - - - - -' POSITION (22,3)
(0036)           FIELD 'ACCOUNT NAME    ' POSITION (10,6)
(0037)    NAM    FIELD LENGTH 20 POSITION (35,6) NOPROTECT
(0038)           FIELD '    AND ADDRESS' POSITION (10,7)
(0039)    AD1    FIELD LENGTH 20 POSITION (35,7) NOPROTECT
(0040)    AD2    FIELD LENGTH 20 POSITION (35,8) NOPROTECT
(0041)    AD3    FIELD LENGTH 20 POSITION (35,9) NOPROTECT
(0042)           FIELD 'STOCK NUMBER' POSITION (10,12)
(0043)    STKNO  FIELD LENGTH 6 POSITION (35,12) NOPROTECT
(0044)           FIELD 'QUANTITY' POSITION (10,14)
(0045)    STKQTY FIELD LENGTH 5 POSITION (35,14) NOPROTECT
(0046)           FIELD 'DELIVERY DATE' POSITION (10,16)
(0047)    STKDEL FIELD LENGTH 8 POSITION (35,16) NOPROTECT
(0048)           FIELD 'ORDER REFERENCE IS' POSITION (10,20)
(0049)    ONO    FIELD LENGTH 5 POSITION (35,20)
(0050)           END DEVICE
(0051)           END FORMAT

0000 ERRORS (FDL, REV 16 - 16-FEB-79)
```

```
 D E V I C E   F O R M A T   M A P      FORMAT: DS1       DEVICE: VISTAR3    SIZE: 24 BY 80    PAGE: 1
Ø
Ø
       ....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*....8
      ----------------------------------------------------------------------------------
Ø  1 |                                                                                  |
Ø  2 |                    O R D E R   E N T R Y   S Y S T E M                          |
Ø  3 |                    - - - - -   - - - - -   - - - - - -                           |
Ø  4 |                                                                                  |
Ø  5 |                                                                                  |
Ø  6 |         ACCOUNT NAME                                                             |
+
Ø  7 |             AND ADDRESS         ____________________                             |
+
Ø  8 |                                 ____________________                             |
+
Ø  9 |                                 ____________________                             |
+
Ø 10 |                                 ____________________                             |
Ø 11 |                                                                                  |
Ø 12 |         STOCK NUMBER                                                             |
+
Ø 13 |                                 ______                                           |
Ø 14 |         QUANTITY                                                                 |
+
Ø 15 |                                 _____                                            |
Ø 16 |         DELIVERY DATE                                                            |
+
Ø 17 |                                 ________                                         |
Ø 18 |                                                                                  |
Ø 19 |                                                                                  |
Ø 20 |         ORDER REFERENCE IS      *****                                            |
Ø 21 |                                                                                  |
Ø 22 |                                                                                  |
Ø 23 |                                                                                  |
Ø 24 |                                                                                  |
Ø     ----------------------------------------------------------------------------------
```

```
 D E V I C E   F O R M A T   M A P      FORMAT: DS1       DEVICE: OWL1200     SIZE: 24 BY 80    PAGE: 1
Ø
Ø
        ....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*....8
       ----------------------------------------------------------------------------------
Ø  1 |                                                                                  |
Ø  2 |                    O R D E R   E N T R Y   S Y S T E M                           |
Ø  3 |                    - - - - -   - - - - -   - - - - - -                           |
Ø  4 |                                                                                  |
Ø  5 |                                                                                  |
Ø  6 |        ACCOUNT NAME                                                              |
+                                     ____________________
Ø  7 |            AND ADDRESS                                                           |
+                                     ____________________
Ø  8 |                                                                                  |
+                                     ____________________
Ø  9 |                                                                                  |
+                                     ____________________
Ø 10 |                                                                                  |
Ø 11 |                                                                                  |
Ø 12 |        STOCK NUMBER                                                              |
+                                     ______
Ø 13 |                                                                                  |
Ø 14 |        QUANTITY                                                                  |
+                                     _____
Ø 15 |                                                                                  |
Ø 16 |        DELIVERY DATE                                                             |
+                                     ________
Ø 17 |                                                                                  |
Ø 18 |                                                                                  |
Ø 19 |                                                                                  |
Ø 20 |        ORDER REFERENCE IS        *****                                           |
Ø 21 |                                                                                  |
Ø 22 |                                                                                  |
Ø 23 |                                                                                  |
Ø 24 |                                                                                  |
Ø      ----------------------------------------------------------------------------------
```

APPENDIX I

ADVANCED USE OF FORMS

This appendix gives an example of FORMS used in conjunction with an external login program. (Refer to the Systems Administrator's Guide PDR3109). This program should be installed locally under the supervision of the system administrator. It is a good example of how forms can be used to produce a menu-driven approach to operation, and it is also a good example of how to use COBOL to produce an external login program.

The program is listed on the following pages.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    DEMO.
*
*   THIS COBOL PROGRAM WAS DEVELOPED WITH THE PRIME FORMS
*   PACKAGE TO DEMONSTRATE THE ABILITY TO FORMAT SCREENS
*   WITH A 'MENU' APPROACH TO OPERATOR INTERACTION.
*
*   THE SCREEN DEFINITIONS ARE CONTAINED IN THE FILES
*           LOGINO1         FIRST MENU
*           LOGINO2         SECOND MENU
*           LOGINO3         PASSWORD VALIDATION SCREEN
*
*   THE ONLY 'TRICKY' CODE IN THE PROGRAM IS THE SPECIAL
*   PROGRAMMING NECESSARY TO ALLOW THIS CODE TO BE SAVED
*   AS AN R-MODE VERSION IN CMDNCO KNOWN BY THE NAME
*    'LOGIN'
*   THE R-MODE COMPILE OF THIS PROGRAM USES THE STOP LITERAL.
*   THE V-MODE VERSION USES THE 'EXIT PROGRAM' IN A DUMMY
*   PARAGRAPH TO FAKE THE COMPILER INTO THINKING THAT THIS
*    IS A COBOL SUBROUTINE - HENCE NO FILE ASSIGNMENTS - EVER.
*
*   THE R-MODE SEQUENCE IS TO R *COB. FORM
*                                ENTER A SLASH FOR FILE ASSIGNMENTS
*                                AND SAVE THE RESULT WITH 1/177777
*   THE V-MODE VERSION MUST BE SEG'ED AND THEN RUN THROUGH
*   THE SEG UFD COMMAND PROCEDURE 'CO CMDSEG' THAT WILL MAKE IT
*   LOOK LIKE THE R-MODE COUNTERPART.  THENCE OFF TO CMDNC0 AS 'LOGIN'.
*
*   THE PRIME SYSTEM SUBROUTINE 'TIMDAT' CAN BE CALLED
*   FROM THIS PROGRAM MODULE TO HAVE ACCESS TO LOGIN
*   SESSION RESOURCE CONSUMPTION DATA.
*   THIS USER DEPENDENT DATA MIGHT BE WRITTEN OUT TO A
*   DISK JOB ACCOUNTING FILE FOR FURTHER PROCESSING.
*
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FORM ASSIGN TO TERMINAL.
*
*
*
 DATA DIVISION.
 FILE SECTION.
*
*
 FD FORM LABEL RECORDS ARE OMITTED.
 01  FORM-RECORD                PIC X(40).
*
*
 WORKING-STORAGE SECTION.
 01  INVOKE-COMMAND.
     05  INVOKE-LITERAL   PIC X(9)   VALUE  '##INVOKE'.
```

```
    05  INVOKE-FORM-NAME PIC X(8)    VALUE  SPACES.
 01  RELEASE-COMMAND      PIC X(9)    VALUE  '##RELEASE'.
 01  CLEAR-COMMAND        PIC X(7)    VALUE  '##CLEAR'.
 01  CLEAR-ALL-COMMAND    PIC X(11)   VALUE  '##CLEAR ALL'.
 01  FUNCTION-KEY-ENABLE  PIC X(10)   VALUE  '##FKEYS ON'.
 01  FUNCTION-KEY-RETURN   PIC X(3).
 01  ABSOLUTE-POSITION   PIC X(17) VALUE '##POSITION SELECT'.
 01  PASSWORD-RESPONSE       PIC X(8)  VALUE SPACES.
 01  COMMAND-LINE.
    05  LOGOUT-COMMAND   PIC X(6).
    05  ABBREVIATED-LO REDEFINES LOGOUT-COMMAND.
        10  FILLER           PIC X(2).
        10  SHORT-LOGOUT     PIC X(2).
        10  FILLER        PIC X(2).
    05  FILLER            PIC X(30).
*
*
*
 PROCEDURE DIVISION.
 PARAGRAPH-01.
*    STOP 'TEMPORARY HALT'.
*     THE STOP  LITERAL IS USED FOR R-MODE PGMS
*     TO ALLOW PROGRAM SAVING AFTER THE FILE ASSIGNMENTS
*
*     THE CMDREAD CALL IS USED TO ALLOW THE USE OF THIS
*     PROGRAM AS A SYSTEM WIDE LOGIN ROUTINE
*     IT TRAPS THE LOGOUT COMMANDS TO INSURE THAT THE USER
*     DOESN'T GET A MENU AT LOGOUT TIME !!
*     CALL 'CMREAD' USING COMMAND-LINE.
*     IF LOGOUT-COMMAND EQUAL 'LOGOUT' OR
*          SHORT-LOGOUT EQUAL '  '
*                    STOP RUN
*     ELSE
*              NEXT SENTENCE.
*
*
      OPEN I-O FORM.
*
*     PREPARE FIRST SCREEN MENU
      MOVE 'LOGIN01 ' TO INVOKE-FORM-NAME.
      WRITE FORM-RECORD FROM INVOKE-COMMAND.
      WRITE FORM-RECORD FROM FUNCTION-KEY-ENABLE.
      WRITE FORM-RECORD FROM ABSOLUTE-POSITION.
*
*
*
 PARAGRAPH-02.
     READ FORM INTO FUNCTION-KEY-RETURN.
     IF FUNCTION-KEY-RETURN EQUAL ' 06'
               GO TO PARAGRAPH-03
     ELSE
              GO TO PARAGRAPH-99.
*
```

```
*
*
 PARAGRAPH-03
     WRITE FORM-RECORD FROM CLEAR-ALL-COMMAND.
     WRITE FORM-RECORD FROM RELEASE-COMMAND.
*    PREPARE THE SECOND MENU SCREEN
     MOVE 'LOGIN02' TO INVOKE-FORM-NAME.
     WRITE FORM-RECORD FROM INVOKE-COMMAND.
     WRITE FORM-RECORD FROM FUNCTION-KEY-ENABLE.
     WRITE FORM-RECORD FROM ABSOLUTE-POSITION.
*
*    OK..NOW GET THE FUNCTION KEY RETURN
     READ FORM INTO FUNCTION-KEY-RETURN.
     IF FUNCTION-KEY-RETURN EQUAL '  04'
               GO TO PARAGRAPH-04
     ELSE
               GO TO PARAGRAPH-99.
*
*
*
 PARAGRAPH-04.
     WRITE FORM-RECORD FROM CLEAR-ALL-COMMAND.
     WRITE FORM-RECORD FROM RELEASE-COMMAND.
     MOVE 'LOGIN03' TO INVOKE-FORM-NAME.
     WRITE FORM-RECORD FROM INVOKE-COMMAND.
     READ FORM INTO PASSWORD-RESPONSE.
     IF PASSWORD-RESPONSE NOT EQUAL 'TB
                  GO TO PARAGRAPH-99
     ELSE
                  GO TO PARAGRAPH-98.
*
*
*
 PARAGRAPH-98.
*     HOOK HERE FOR ACCOUNTING DATA
*    OR PASSWORD RECORDS ETC..
     WRITE FORM-RECORD FROM CLEAR-ALL-COMMAND.
     WRITE FORM-RECORD FROM RELEASE-COMMAND.
     CLOSE FORM.
     DISPLAY '****  TRIAL BALANCE APPLICATION INVOKED HERE ****'.
      STOP RUN.
*
*
*
 PARAGRAPH-99.
*    HOOK HERE FOR ACCOUNTING FILE INFO
*     CALL 'TIMDAT' AND RECORD FUNCTION SEQUENCE AND PW
     WRITE FORM-RECORD FROM CLEAR-ALL-COMMAND.
     WRITE FORM-RECORD FROM RELEASE-COMMAND.
     CLOSE FORM.
     DISPLAY 'FUNCTION ERROR TRAP INVOKED'.
     STOP RUN.
 PARAGRAPH-DUMMY.
```

EXIT PROGRAM.

The forms descriptor associated with this program is as follows:

```
LOGIN01      STREAM
                 FIELD    SELECT   LENGTH 1
                 FIELD        (DATE3,DATE3)
                 FIELD        (TIME1,TIME1)
             END STREAM
*
*
*
LOGIN01   FORMAT
          DEVICE   CWL1200
          FIELD  '  I C P   P350   C O N T R O L   C E N T E R';
                          POSITION (15,2) NOPROTECT
DATE3     FIELD LENGTH 8 POSITION (25,4) NOPROTECT
TIME1     FIELD LENGTH 5 POSITION (47,4) NOPROTECT
          FIELD  'F1  ACCOUNTS RECEIVABLE' POSITION (25,7)
          FIELD  'F2  ACCOUNTS PAYABLE'    POSITION (25,9)
          FIELD  'F3  WORD PROCESSING'     POSITION (25,11)
          FIELD  'F4  PROGRAM DEVELOPMENT'  POSITION (25,13)
          FIELD  'F5  SYSTEM ADMINISTRATOR' POSITION (25,15)
          FIELD  'F6  GENERAL LEDGER'       POSITION (25,17)
          FIELD 'SELECT FUNCTION KEY ACTIVITY' POSITION (5,21);
                        NOPROTECT BLINK
SELECT    FIELD  LENGTH 1                   POSITION (38,21) NOPROTECT
          END DEVICE
          END FORMAT
```

```
LOGIN02  STREAM
         FIELD SELECT LENGTH 1
         END STREAM
*
*
*
LOGIN02  FORMAT
         DEVICE OWL1200
         FIELD 'G E N E R A L   L E D G E R' POSITION (25,4);
                    NOPROTECT
         FIELD 'F1  BUDGET RATIO ANALYSIS' POSITION (25,9)
         FIELD 'F2  JOURNAL'                POSITION (25,12)
         FIELD 'F3  PROFIT   LOSS STATEMENT' POSITION (25,15)
         FIELD 'F4  TRIAL BALANCE'          POSITION (25,18)
         FIELD  'SELECT FUNCTION KEY ACTIVITY'     POSITION (5,21);
                    NOPROTECT BLINK
SELECT FIELD LENGTH 1 POSITION (38,21) NOPROTECT
           END DEVICE
           END FORMAT



LOGIN03   STREAM
           FIELD PASSWD LENGTH 8
          END STREAM
*
*
LOGIN03  FORMAT
          DEVICE OWL1200
          FIELD 'VALIDATE YOUR TRIAL BALANCE ACCESS PASSWORD'  ;
                        POSITION (10,12)
PASSWD     FIELD LENGTH 8 POSITION (58,12) NOPROTECT NODISPLAY
          END DEVICE
          END FORMAT
```

APPENDIX J

ERROR MESSAGES

ERROR MESSAGE FORMAT

All errors generated by the FDL translator and FAP are of the form:

    C#nn   text message

Where nn represents a unique two-digit error code for each type of error. The message printed is a one-line diagnostic of the cause of the error and possibly what action has been taken by the translator. The following paragraphs describe the error codes generated by FDL. Unless otherwise indicated, the statement that caused an error is ignored by FDL.

FDL ERROR MESSAGES

The following paragraphs list FDL error messages and explanations.

C#00 BAD STATEMENT FORMAT

The contents of the statement field is not an alphanumeric text item.

C#01 STATEMENT NOT RECOGNIZED.

The statement field does not contain a valid FDL statement.

C#02 ARGUMENT REQUIRED.

An argument is required following the statement name.

C#03 ARGUMENT TOO LONG.

A text item exceeds 80 characters in length.

C#04 MULTIPLY DEFINED MACRO.

A macro by the same name already exists. This statement is ignored and the previous macro definition is retained.

C#05 BAD NAME FIELD.

The name field (starting in the left margin) contains an illegal character.

C#06 NAME REQUIRED.

A name must be present in the name field (starting in the left margin). This error is generally issued because a mapped field in the FORMAT descriptor is missing a name.

C#07 STATEMENT FIELD IS BLANK.

A name was present in the name field, but no statement followed.

C#08 NO END STATEMENT; END ASSUMED

An end-of-file was encountered while processing a STREAM or FORMAT descriptor. An END STREAM or END FORMAT is assumed.

C#09 NOT PROCESSING STREAM DESCRIPTOR.

An END STREAM or SUBSTREAM statement was issued and a stream descriptor is not being processed.

C#10 END SUBSTREAM MISSING. IT IS ASSUMED HERE.

An END STREAM statement was issued while a substream block was being processed. An END SUBSTREAM is assumed prior to the END STREAM.

C#11 NOT PROCESSING SUBSTREAM

An END SUBSTREAM statement was issued while not processing a substream block.

C#12 NOT PROCESSING FORMAT

An END FORMAT or a DEVICE statement was issued while not processing a FORMAT descriptor.

C#13 END DEVICE MISSING. IT IS ASSUMED HERE.

An END FORMAT was encountered while still processing a device description. An END DEVICE is generated prior to the END FORMAT.

C#14 NOT PROCESSING DEVICE BLOCK.

A FIELD definition was issued after a FORMAT statement, but before a DEVICE block was started.

C#15 END STATEMENT MISSING; IT IS ASSUMED HERE.

A STREAM or FORMAT descriptor was not terminated before another was started. An END STREAM or END FORMAT is generated prior to this statement.

C#17 BAD PARAMETER.

This indicates that an unrecognizable parameter was present on a FIELD statement.

C#18 INVALID FORMAT NAME.

The name supplied following the FORMAT parameter in the STREAM statement does not conform to the naming conventions discussed earlier in this document.

C#19 NAME NOT PERMITTED.

A name was present on a statement which does not permit one. This usually means that a literal field in the FORMAT descriptor contains a name.

C#21 ALREADY PROCESSING SUBSTREAM.

A SUBSTREAM statement was issued while already processing a substream block.

C#22 VALIDATION STRING MISSING.

The VALIDATE parameter is present on a STREAM descriptor field, but is not followed by any validation masks.

C#23 BAD JUSTIFY PARAMETER.

The JUSTIFY parameter in the FIELD statement is not followed by one of its four valid arguments.

C#24 MAPPING SPECIFICATION REQUIRED.

A STREAM descriptor FIELD is not followed by any mapping specification.

C#25 BAD MAPPING SPECIFICATION.

A STREAM descriptor FIELD is not followed by a valid mapping specification.

C#26 BAD LENGTH SPECIFICATION.

The LENGTH parameter in either STREAM or FORMAT descriptor is not followed by a valid numeric argument.

C#27 BAD INPUT-OUTPUT SPECIFICATION.

An INPUT, OUTPUT, or INPUT-OUTPUT parameter has been misused. This usually means that INPUT-OUTPUT or OUTPUT has been issued when processing an input-literal field.

C#28 MAP FIELD NAME TOO LONG.

The map to field name in a STREAM descriptor FIELD is longer than eight characters.

C#29 ALREADY PROCESSING DEVICE BLOCK.

A DEVICE statement has been issued while already processing a device block.

C#30 SYNTAX ERROR.

This general error message is issued whenever two items in a field definition are separated by an illegal character.

C#31 BAD POSITION PARAMETER.

The POSITION parameter in a FORMAT descriptor FIELD is not followed by a valid argument.

C#32 POSITION OUT OF RANGE.

One or more of the arguments in the POSITION parameter is zero.

C#33 LENGTH PARAMETER MISSING.

The length declaration for a STREAM or FORMAT descriptor FIELD is required but not supplied.

C#34 POSITION PARAMETER MISSING.

The POSITION parameter in a FORMAT descriptor FIELD is not supplied.

C#35 UNRECOGNIZED SYSTEM INFORMATION FIELD NAME.

The name specified in a System Information Field is unrecognized.

C#36 INPUT/OUTPUT SPECIFICATION NOT PERMITTED.

An INPUT, OUTPUT, or INPUT-OUTPUT specification was included on a system information field definition.

C#37 UNRECOGNIZED PARAMETER.

See C#17.

C#38 NOT PROCESSING STREAM/DEVICE FORMAT BLOCK.

A field definition has been issued outside of a STREAM or FORMAT descriptor. This and all other FIELD declarations up to the next STREAM, FORMAT, or DEVICE statement are ignored. This error message is issued once per each violation.

C#39 MULTIPLY DEFINED SYMBOL.

A FIELD name has been redefined within the same STREAM or FORMAT descriptor. This field is processed normally, but will produce undesired results at run-time.

C#40 BAD START SPECIFICATION.

The argument following the START specification in the FIELD definition within a STREAM is not numeric and greater than zero.

C#41 ILLEGAL MACRO ARGUMENT SPECIFIER.

The item following the argument reference symbol (#) is not numeric and greater than zero.

C#42 EOF ENCOUNTERED BEFORE END REPEAT.

An end-of-file was encountered on the input file before a repeat block was terminated. This usually causes abortion of the translation.

C#43 END REPEAT MISSING - REPEAT BLOCK IGNORED.

An END statement was encountered while processing a REPEAT block. The entire REPEAT block is ignored and the END statement processed.

C#44 STATEMENT NOT ALLOWED WITHIN REPEAT BLOCK.

A statement other than a FIELD statement was found within a REPEAT block. The statement is ignored; processing of the REPEAT block continues.

C#45 INPUT/OUTPUT SPECIFICATION REQUIRED.

An input/empty-condition or output-literal field did not contain a required INPUT or OUTPUT statement.

C#46 INCONSISTENT SUBSTREAM USAGE.

A FIELD definition appears outside of a SUBSTREAM block in a multi-record stream definition -or- the user has attempted to start a SUBSTREAM definition when previously defined FIELDS do not reside within a SUBSTREAM. This error message is only issued once per STREAM descriptor.

FAP ERROR MESSAGES

Like FDL error messages, all FAP error messages are of the form:

t#nn text message

The t in the error code represents the error type. At present, there are three types:

- F - file system/input file/control block error
- S - syntax error
- T - TCB or DCF format error

nn represents a two-digit error number, unique for each error message generated by FAP.

The following paragraphs list FAP error messages and explanations.

F#00 CONTROL BLOCK UFD DOES NOT EXIST.

An operation other than CREATE was attempted and the forms UFD ('FORMS*') does not exist on the system.

F#01 CONTROL BLOCK DIRECTORY DOES NOT EXIST.

An operation other than CREATE was attempted and the FORMS segment directory ('FMS.**') does not exist within the FORMS UFD.

F#04 INPUT FILE IS EMPTY.

The input file specified in an ADD or REPLACE command is empty.

F#05 PREMATURE EOF.

An EOF was encountered on the input file in an ADD or REPLACE command before the end-of-data record. The module is deleted from the control directory. This is usually caused by the user pressing the BREAK key in the middle of an FDL compilation.

F#06 FILE DOES NOT EXIST.

The input file specified in an ADD or REPLACE command does not exist in the current UFD.

F#07 BAD INPUT FILE.

The input file specified in an ADD or REPLACE command is not a valid FDL output binary file. No action is taken with this file.

F#08 I/O LIST OVERFLOW, LINK SUPPRESSED.

FAP ran out of room while attempting to link a STREAM definition to a FORMAT an out of room while attempting to link a STREAM definition to a FORMAT definition. The internal I/O buffer must be enlarged before this form definition may be added. Increase the value of IOLSIZ in the $INSERT file FORMS>FAP>IOBUF$ and rebuild FAP.

F#09 STREAM/FORMAT BUFFER OVERFLOW.

FAP ran out of room attempting to read a STREAM or FORMAT descriptor binary ran out of room attempting to read a STREAM or FORMAT descriptor binary file. The buffer must be enlarged and FAP rebuilt before this form definition can be added. Increase the value of SFBSIZ in the $INSERT file FORMS>FAP>IOBUF$ and rebuild FAP. >FAP>IOBUF$ and rebuild FAP.

F#10 ERROR READING STREAM / FORMAT DESCRIPTION.

A file system error occurred while attempting to load a STREAM or FORMAT descriptor.

F#11 ERROR READING / DELETING LINK FILE.

A file system error occurred when FAP was trying to purge a linked form definition file.

F#12 ERROR RENAMING LINK FILE.

An error occurred when FAP attempted to rename a link file following a PURGE operation.

S#00 FILE NAME REQUIRED.

An ADD or REPLACE command was issued, but no file name followed. The command is ignored.

S#01 BAD FORM NAME SPECIFIER.

The form name specifier contained a syntax error. This command is ignored.

S#02 BAD ARGUMENT.

One of the parameters in the command line was not recognized. The command is ignored.

S#03 BAD TYPE.

The form name specifier contained a type declaration other than STR (stream) or FMT (format). This command is ignored.

S#04 NO FORM NAME SPECIFIED.

A PURGE command was issued without a required form name specifier. The PURGE command is ignored.

S#05 MISSING ARGUMENT.

The TCB command was issued without any following user number. The command is ignored.

S#06 BAD USER NUMBER.

The user number specified in the TCB command is not an integer number greater than zero. The TCB command is ignored.

S#07 BAD TERMINAL NAME.

The user attempted to assign the name PRINTER as a terminal type in a TCB command. This is not permitted and the TCB command is ignored.

T#00 DCF DEVICE INTERLUDE FIELD ERROR.

The device interlude number field in the given DCF entry is not numeric or greater than zero. The DCF must be edited and corrected before continuing.

T#01 DCF DEVICE NAME FIELD ERROR.

The device name field in the given DCF entry contains an illegal character or is empty. The DCF must be edited and corrected before continuing

T#02 DCF DEVICE ABBREVIATION FIELD ERROR.

The device abbreviation field in the given DCF entry is empty or contains a space or illegal character. The DCF must be edited and corrected before continuing.

T#03 TCB LINE/COLUMN FIELD ERROR.

The line or column specification field in the given DCF entry is empty, contains a non-numeric value, or is less than 1. The DCF must be edited and corrected before continuing.

T#04 MAX DEVICE NUMBER EXCEEDED IN DCF.

The DCF contains an entry with a device interlude number greater than fifty (50). This error is issued from the GENERATE command only. Only fifty (50) (!) devices may be in use at one time.

T#05 DEVICE CONTROL FILE EMPTY.

The DCF is empty and the user issued a TCB or GENERATE command.

T#06 TERMINAL UNDEFINED.

The terminal type specified in the TCB command is not present in DCF.

##, prefix for run-time directives 3-1

$INSERT files 5-19

ADD command 6-1

Advanced use of FORMS I-1

Alternate input file 5-19

Application program, compiling 7-18, 8-13

Application program, loading 7-18, 8-14

Applications programs, FORMS, how to write 2-1

Attribute modification directives 4-8

BLINK parameter 5-15

Bug chart D-1

CLEAR, run-time directive 4-3

COBOL program example, form descriptor for H-1

COBOL program listing, sample G-1

Compiling the application program 7-18, 8-13

Configurable I/O list 4-10

Configuration file, terminal B-2

Control file, device B-2

CREATE command 6-2

Creating the form descriptor file 7-11, 8-9

Data areas, setting up 7-13, 8-10

Data description, purpose of 3-1

Data format 3-2

Data stream descriptor 3-2

Default options 5-21

DEFINE statement 5-17

Defining device types 7-12, 8-10

Descriptor structure 5-2

Descriptor, data stream 3-2

Descriptor, device format 3-3

Descriptor, field 3-2

Device control file B-2

Device definition database B-2

Device definition statements 5-5

Device driver, offline printer B-4

Device driver, Owl B-5

Device driver, VISTAR/3 B-4

Device drivers, installing C-6

Device drivers, Prime supplied B-3

Device drivers, user written C-1

Device format descriptor 3-3

Device I/O B-1

Device I/O mechanism B-1

Device input/output system B-1

Device mapping scheme B-3

DEVICE statement 5-5

Device types, defining 7-12, 8-10

Direct field 5-6

Directives, attribute modification 4-8

Directory information A-1

Display atrtribute Parameters 5-15

DISPLAY parameter 5-16

EJECT statement 5-19

END DEVICE statement 5-5

END FORMAT statement 5-5

END REPEAT statement 5-18

END STREAM statement 5-4

END SUBSTREAM statement 5-4

Error handling, run-time 4-9

Error messages J-1

Example COBOL program 8-1

Example FORTRAN program 7-1

FAP command format 6-1

FAP example 6-9

FAP functions 1-6

FAP overview 1-6

FAP see also Forms Administrative Processor

FAP, see FORMS Administrative Processor

FDL 5-1

FDL command 3-7

FDL listing 3-9

FDL listing details, specifying 7-16, 8-12

FDL options 3-9

FDL source, translating 7-18, 8-13

FDL syntax 5-1

FDL temporary files 5-22

FDL translation command format 5-20

Field definition 3-4

FIELD definition 3-4

Field definition examples 5-13, 5-16

Field descriptor 3-2

Field generation, iterative 5-17

FIELD parameter for FORMAT 5-14

FIELD parameters 5-8

Field statement differences between FIELD and STREAM descriptors 3-7

Field statements within a format descriptor 5-14

Field statements within a stream descriptor 5-6

FIELD types 5-6, 5-14

File Description Language command 3-7

File handling, run-time 4-8

Filler field 5-7

FIX parameter 5-11

FKEYS, run-time directive 4-1

FORCEREAD, run-time directive 4-4

Form definition delimiter statements 5-2

Form definition overview 2-1

Form definition summary 3-3

Form definitions 3-1

Form description, setting up 7-11, 8-9

Form descriptor file, creating 7-11, 8-9

Form descriptor for COBOL program example H-1

Form descriptor for FORTRAN program example F-1

Form descriptor preparation 3-5

Form, how to define 3-1

Format coding 3-5

FORMAT descriptor 3-2

Format descriptor boundaries 3-5

Format of data 3-2

FORMAT statement 5-5

Forms Administrative Processor see also FAP

FORMS Administrative Processor, see FAP

FORMS Administrative Processor 6-1

Forms Catalog, installing form descriptor 7-18, 8-14

FORMS Definition Catalog overview 1-6

Forms Definition Language 5-1

FORMS directives summary 2-5

FORMS directives, definitions 2-3

FORMS run-time directives 4-1

FORMS, advantages of 1-2

FORMS, components of 2-3

FORMS, purpose of 1-2

FORMS, using overview 2-3

FORTRAN program example form descriptor F-1

FORTRAN program, sample E-1

FREE parameter 5-16

Functional overview, FORMS and PRIMOS 1-4

GENERATE command 6-3

HOLD parameter 5-16

I/O statement, FORMS, how to write 2-1

I/O, list, configurable 4-10

Input data 7-14, 8-12

Input empty conditional field 5-7

Input literal field 5-6

INPUT parameter 5-9

INPUT-OUTPUT parameter 5-9

Installing FORMS A-1

INVOKE, run-time directive 4-5

IOCS interlude B-1

Iterative field generation 5-17

JOURNAL command 6-4

JUSTIFY parameter 5-8, 5-15

Language manuals, related 1-7

Languages, application, for FORMS 1-5

LENGTH parameter 5-8, 5-15

LINK command 6-4

LIST command 6-5

Listing control statements 5-19

Listing details, FDL, specifying 7-16, 8-12

Listing features 7-17

Literal data 7-14, 8-12

Literal field 5-14

Loading the shared library 4-10

Macro definition 5-17

Macro definitions, using 7-15

Manual, using 1-1

Manuals, related 1-7

Mapped field 5-14

Mapping scheme, device B-3

Mapping stream and format descriptor fields 3-4

MIDAS file template 8-15

Naming conventions 5-2

NOBLINK parameter 5-16

NODISPLAY parameter 5-16

NOFIX parameter 5-11

NOLIST statement 5-19

NOPROTECT attribute, using 3-5

NOPROTECT parameter 5-15

NORMAL VIDEO parameter 5-16

Offline printer device driver B-4

Operating system and utilities manuals, related 1-7

Output data 7-14, 8-12

Output literal field 5-6

OUTPUT parameter 5-9

Overview of FORMS 1-1

Overview, manual 1-1

Owl device driver B-5

POSITION parameter 5-15

Position parameter, relative 5-18

POSITION, run-time directive 4-5

PRIMOS interfaces 1-3

PRINT, run-time directive 4-5

Problem solving chart D-1

Program, running 7-19, 8-16

Programming aids 5-16

PROTECT parameter 5-15

PURGE command 6-5

QUIT command 6-7

Related manuals 1-7

Relative position parameter 5-18

RELEASE, run-time directive 4-6

REPEAT statement 5-18

Repeated text 7-16

REPLACE command 6-7

REVERSE VIDEO parameter 5-16

Run-time directives, FORMS 4-1

Run-time directives, using 4-1

Run-time error handling 4-9

Run-time file handling 4-8

Run-time message 5-22

Running the program 7-19, 8-16

Sample COBOL program listing G-1

Sample FORTRAN program E-1

Setting up a form description 7-11, 8-9

Shared library, loading 4-10

SPACE-FILL parameter 5-9

START parameter 5-12

Stream coding 3-7

STREAM definition statements 5-4

STREAM descriptor 3-2

STREAM statement 5-4

SUBSTREAM statement 5-4

SUBSTREAM, run-time directive 4-6

SUBSTREAMS, using 7-12

Syntax, FDL 5-1

System informational field 5-7

TCB command 6-7

Template, MIDAS file 8-15

Temporary files, FDL 5-22

Terminal configuration file B-2

Text, repeated 7-16

Translating FDL source 7-18, 8-13

Trouble shooting D-1

Using the manual 1-1

VALIDATE parameter 5-10

VALIDATE, run-time directive 4-6

Variable data 7-14, 8-12

VISTAR/3 device driver B-4

ZERO-FILL parameter 5-9