60460430

CONTROL DATA CORPORATION

NOS VERSION 2
SCREEN FORMATTING
REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
CYBER 170
MODELS 815,825,835,845,855
CYBER 180
MODELS 810,830,835,845,855
CYBER 70
MODELS 71,72,73,74
6000

60460430

CONTROL DATA CORPORATION

NOS VERSION 2 SCREEN FORMATTING REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:

CYBER 170
  MODELS 815,825,835,845,855
CYBER 180
  MODELS 810,830,835,845,855
CYBER 70
  MODELS 71,72,73,74
6000

REVISION RECORD

REVISION	DESCRIPTION
A (10-11-83)	Manual released. This revision reflects NOS 2.2 at PSR level 596.
B (10-05-84)	This manual reflects NOS 2.3 at PSR level 617. This revision includes the terminal definition utility (TDU) which provides the capability to define display terminals to be used in screen mode. PDU now supports Pascal programs. This revision also includes the following system-defined terminals: CDC 722, Tektronix 4115, Zenith Z19/Heathkit H19, DEC VT100, and Lear Siegler ADM3A and ADM5. Due to extensive changes, change bars and dots are not used, and all pages reflect the current revision level. This edition obsoletes all previous editions.
Publication No. 60460430	

REVISION LETTERS I, O, Q, S, X AND Z ARE NOT USED.

Address comments concerning this
manual to:

Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of
this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Title Page	-
2	B
3/4	B
5	B
6	B
7	B
8	B
1-1	B
1-2	B
1-3	B
1-4	B
2-1	B
2-2	B
2-3	B
2-4	B
2-5	B
2-6	B
2-7	B
2-8	B
2-9	B
2-10	B
2-11	B
2-12	B
2-13	B
2-14	B
2-15	B
2-16	B
2-17	B
2-18	B
2-19	B
2-20	B
2-21	B
2-22	B
3-1	B
3-2	B
3-3	B
3-4	B
3-5	B
3-6	B
3-7	B
3-8	B
3-9	B
3-10	B
3-11	B
3-12	B
3-13	B
3-14	B
3-15	B
3-16	B
3-17	B
3-18	B
4-1	B
4-2	B
4-3	B
4-4	B
4-5	B
4-6	B
4-7	B
4-8	B
4-9	B
4-10	B
4-11	B
5-1	B
5-2	B
5-3	B
5-4	B
5-5	B
5-6	B
5-7	B
5-8	B
5-9	B
5-10	B
5-11	B
5-12	B
5-13	B
5-14	B
5-15	B
5-16	B
5-17	B
5-18	B
5-19	B
5-20	B
5-21	B
5-22	B
5-23	B
5-24	B
5-25	B
5-26	B
5-27	B
5-28	B
5-29	B
5-30	B
5-31	B
5-32	B
5-33	B
5-34	B
5-35	B
5-36	B
5-37	B
5-38	B
5-39	B
5-40	B
5-41	B
A-1	B
A-2	B
A-3	B
A-4	B
B-1	B
B-2	B
B-3	B
B-4	B
B-5	B
B-6	B
B-7	B
B-8	B
B-9	B
B-10	B
B-11	B
C-1	B
C-2	B
D-1	B
D-2	B
D-3	B
D-4	B
D-5	B
D-6	B
D-7	B
D-8	B
D-9	B
D-10	B
D-11	B
D-12	B
D-13	B
D-14	B
D-15	B
E-1	B
E-2	B
F-1	B
G-1	B
G-2	B
G-3	B
G-4	B
G-5	B
Index-1	B
Index-2	B
Index-3	B
Comment Sheet	B
Back Cover	-

PREFACE

This manual describes the screen formatting feature for the Network Operating System (NOS) Version 2. NOS 2 operates on the CONTROL DATA® CYBER 170 and CYBER 180 Computer Systems.

Programming languages supported by NOS screen formatting are FORTRAN Version 5 and COBOL Version 5 and PASCAL Version 1.1.

The extent to which you can use screen formatting depends on the type of terminal you have. Generally, NOS supports full-screen mode on any display terminal, although some terminals have capabilities that make screen formatting more usable. For more information about these needed capabilities, refer to section 5.

AUDIENCE

This manual is written as a reference for application programmers and NOS procedure writers who want to use the full-screen display capabilities of NOS. For application programmers, this manual assumes a knowledge of FORTRAN 5, COBOL 5, or Pascal 1.1 languages as described in the respective reference manuals. For NOS procedure writers, this manual assumes a knowledge of the structure and use of NOS procedures as described in the NOS 2 Reference Set, Volumes 2 and 3.

ORGANIZATION

This manual is organized according to the major components of the screen formatting feature. The first section gives an overview of NOS screen formatting and its major components. Each of the remaining sections provides a detailed description of one of the components.

The last page of this manual is a comment sheet. Please use this comment sheet to give us your opinion on the manual's usability, to suggest specific improvements, and to report technical or typographical errors. If the comment sheet has already been used, you can mail your comments to:

Control Data Corporation
Publications and Graphics Division ARH219
4201 North Lexington Avenue
St. Paul, Minnesota 55112

Please include the manual title, publication number, and revision level with each inquiry, and indicate whether or not you would like a reply.

CONVENTIONS

Within statement and command format lines, uppercase letters represent words or characters that must be entered exactly as shown. Lowercase letters represent names and values that you supply.

Numbers are assumed to be decimal unless otherwise noted.

In this manual, we refer to the keys as they are labeled on the Viking 721 terminal. Although these are physical keys on the Viking 721, they are also logical keys on other supported terminals. (Refer to appendix G for more information on these keys for the system-defined terminals.) For example, all terminals have an equivalent to the CDC® Viking 721 NEXT key, although the key has different names on different terminals (such as RETURN, NEWLINE, and SEND).

RELATED MANUALS

Readers of this manual may want to refer to one or more of the following manuals.

Control Data Publication	Publication Number
NOS Version 2 Reference Set, Volume 2 Guide to System Usage	60459670
NOS Version 2 Reference Set, Volume 3 System Commands	60459680
FORTRAN Version 5 Reference Manual	60481300
COBOL Version 5 Reference Manual	60497100
Pascal Version 1.1 Reference Manual	60497700

These manuals are available through Control Data sales offices or Control Data Literature Distribution Services (308 North Dale, St. Paul, Minnesota 55103).

DISCLAIMER

This manual describes a subset of the features and parameters documented in Volume 3 of the NOS 2 Reference Set and the programming language reference manuals. Control Data cannot be responsible for the proper functioning of any features or parameters not described in these manuals.

CONTENTS

APPENDIXES

FIGURES

TABLES

INTRODUCTION 1

The NOS screen formatting feature provides full-screen input and output capabilities for NOS procedures and for FORTRAN5, COBOL5, or Pascal 1.1 application programs. This manual describes the modifications and utilities for screen formatting capabilities and how to use them. Procedures require no special modifications to take advantage of full-screen parameter prompting. Existing procedures can be executed in full-screen mode without modification.

Application programs and NOS procedures written to run in full-screen mode can be run on almost any display terminal. The extent to which you can use screen formatting depends on the type of terminal you have. Generally, Control Data supports full-screen mode on any display terminal, although some terminals have capabilities that make screen formatting more usable. For more information about these needed capabilities, refer to section 5. Terminal capabilities and keys can be defined for full-screen use by using the terminal definition utility (TDU). Seven terminals are already system-defined for use of screen formatting and Full Screen Editor (FSE). These terminals are:

- CDC Viking 721
- CDC 722
- Tektronix 4115
- Zenith Z19/Heathkit H19
- DEC VT100
- Lear Siegler ADM3A
- Lear Siegler ADM5

Full-screen displays for application programs are called panels. Panels are stored in user libraries in load capsule format. Your application programs access panels through special screen formatting object routines. A screen definition named in a subroutine read or write operation causes the screen to be displayed at the terminal. Any input or output data that is entered or displayed on the screen is passed between the program and the terminal as parameters of the object routine.

WHAT IS SCREEN FORMATTING?

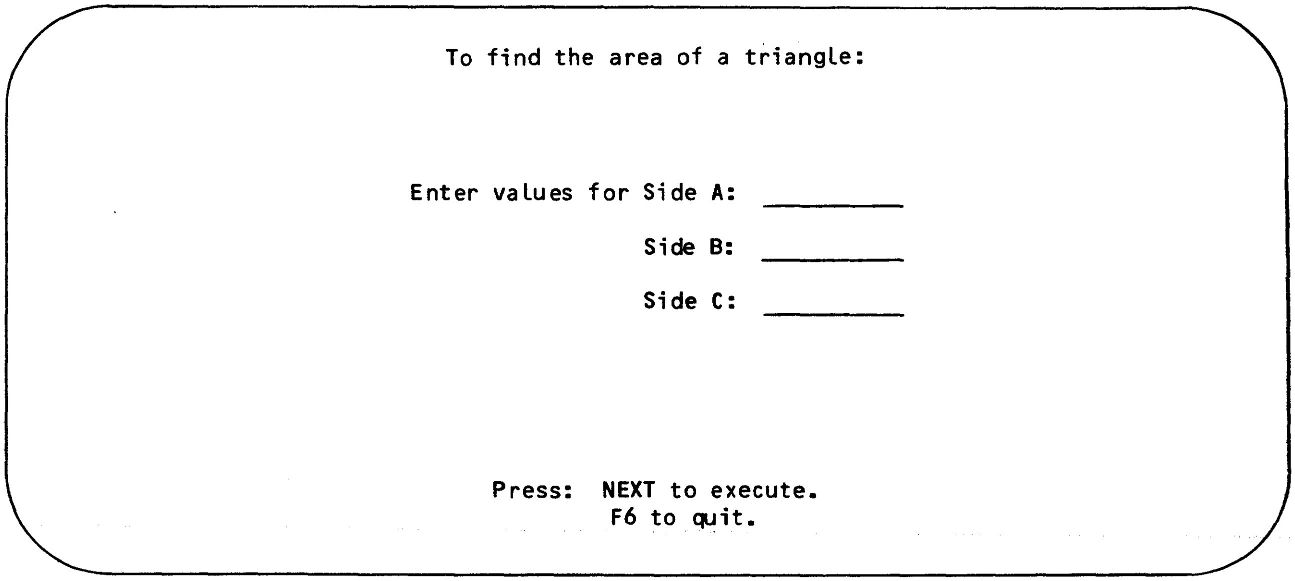
Interactive job processing can be divided into two types: line mode and screen mode. As the name implies, line mode processing handles terminal input and output one line at a time. In response to a system or program prompt, you type in one line of data and submit the line for processing by pressing the NEXT key (carriage return). Pressing the NEXT key sends the line to the CPU for validation checking and execution. If the line contains validation errors, the system prompts you to reenter the line. The system will not display the next prompt until the current line has been properly entered. You can think of line mode entry as a question and answer session in which you must answer each question correctly before moving on to the next one.

In screen mode processing, you are presented with an entire screen of information at one time. The screen can be formatted to display information and to request user input, just as the same information might be formatted on a printed page. Figure 1-1 is an example of a formatted screen.

The screen may contain parameter or variable fields for you to fill in. If so, you may enter the values in any order. To move from one input field to the next, press the tab key (the default entry sequence proceeds from the first field on the screen to the last). The terminal capability that provides tabbing from one input field to the next is called protected tabbing. To enter input in nonsequential order or to modify values entered previously, move the cursor to the fields using the cursor control keys.

You can go back and correct any values entered previously; none of the values are submitted for processing until you are finished with the screen. When satisfied that all entries are correct, press the NEXT key (or another function key, depending on what you have specified in the application) to submit the entire page of data for processing.

When using any terminal that does not support protected tabbing, the tab key must be followed by pressing the key corresponding to NEXT. On these other terminals, you may press the tab key more than once before pressing the NEXT key to position the cursor ahead more than one input field. Any programmable function key will act as a tab key if it is not defined in the panel definition file.



```
                    To find the area of a triangle:



                 Enter values for Side A:  ________

                                  Side B:  ________

                                  Side C:  ________




                      Press:  NEXT to execute.
                              F6 to quit.
```

Figure 1-1. Formatted Screen Display

NOS screen formatting is a set of software tools that makes screen mode display capabilities available to the application programmer and NOS procedure writer. For application programmers, these tools include:

- Utilities and procedures used to create screen definitions and to maintain screen definition libraries.
- Subroutines used to access screen definitions and use them to perform data input and output operations.

SCREEN FORMATTING FOR NOS PROCEDURES

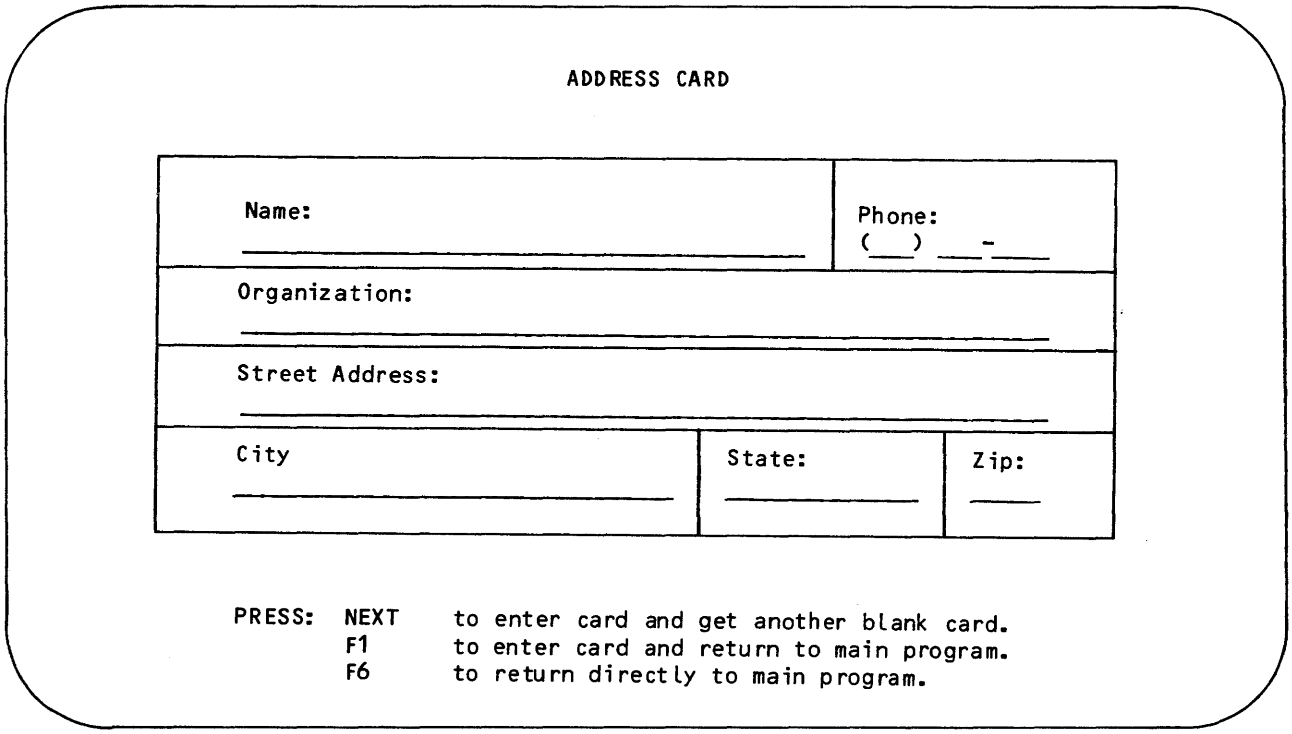
For NOS procedure writers, screen mode parameter display is an automatic system feature. Once you have entered a SCREEN command (as described in the NOS 2 Reference Set, Volume 3), the system automatically presents all subsequent parameter displays in a system-defined, full-screen format. Full-screen parameter display requires no modifications to existing procedure files, although a knowledge of the screen mode formats and features will help you make the most effective use of full-screen display capabilities when writing procedures in the future. The use of NOS procedures in screen mode is described in section 4 of this manual.

SCREEN FORMATTING FOR APPLICATION PROGRAMS

In NOS screen formatting, a full-screen data display used by an application program is referred to as a panel. The panels for a given program are designed by the application programmer. When creating a panel, you can use any type of screen display (such as blank forms, menus, and information display tables) that suits the needs of the program. Using line drawings, you can produce a screen facsimile of a printed form such as the one shown in figure 1-2. If supported on your terminal, you can also incorporate special display features such as blinking characters or inverse video into your panels. (Refer to appendix G for a description of attributes that are available on the system-defined terminals.)

The creation of panels is the function of the panel definition utility (PDU) described in section 2. That section includes a description of the declaration statements and formats used to create a panel definition within an ordinary NOS text file. Also described are two commands used for file maintenance, PDU and ULIB. The PDU command compiles a panel definition file and stores it in a user library, while the ULIB command creates or modifies libraries or library records containing compiled panels.

Once stored in a library, a panel can be accessed by your application program using the screen formatting object routines described in section 3. Each of the object routines performs a specific function related to data input and output at the terminal. These functions include opening and closing panels, reading and writing data using panels, determining the last function key pressed or the last cursor position at which data was entered, and so on.



ADDRESS CARD

Name: ______________________ Phone: (___) ___-____

Organization: ______________________

Street Address: ______________________

City ______________ State: __________ Zip: _____

PRESS: NEXT to enter card and get another blank card.
F1 to enter card and return to main program.
F6 to return directly to main program.

Figure 1-2. Data Entry Panel with Line Drawings

PANEL DEFINITION UTILITY 2

The creation of a panel involves three steps:

- Creation of a panel definition file
- Compilation of the definition file into a load capsule
- Storage of the capsule in a user library

This section explains how to create a panel definition file using a standard NOS text editor and text file. Also described are the PDU and ULIB commands. The PDU command compiles a panel definition file into load capsule form and stores it in a user library. The ULIB command simplifies panel library maintenance tasks.

PANEL DEFINITION FILE

A panel definition file is a NOS 6/12-bit display code text file that describes how a panel appears on the screen, and how user input to the panel is to be handled. It consists of three parts, an optional title line, a declaration section, and an image section. All three parts are contained in the same text file.

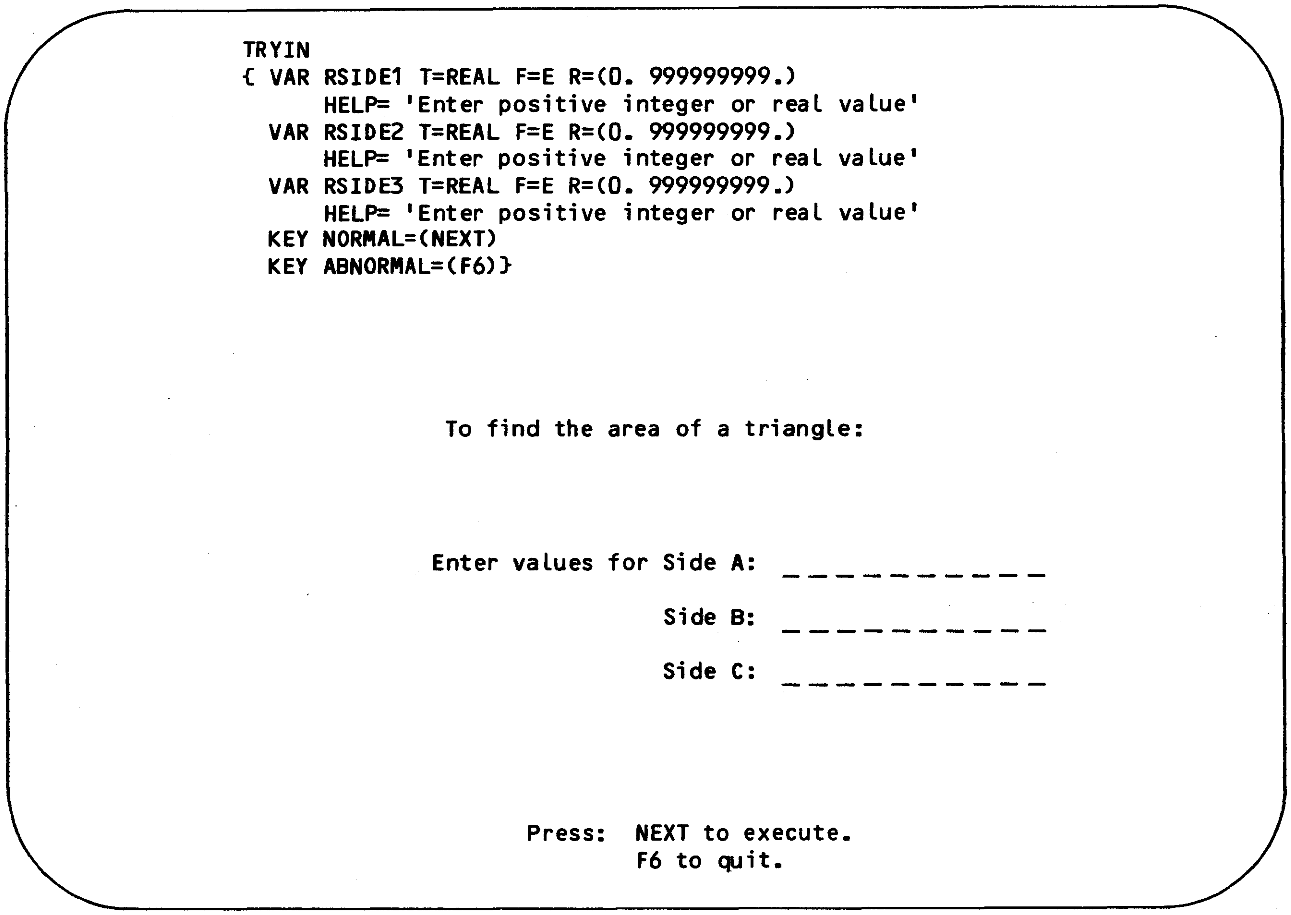
Figure 2-1 shows an example of a panel definition file. The title line contains the file name, TRYIN. The declaration section follows the title line and consists of a block of definition statements enclosed in braces. The image section consists of all file lines following the last line of the declaration section. Figure 2-2 shows the panel produced by the file in figure 2-1.

A panel definition file image can contain up to 64 lines of up to 160 columns. The amount of material that will be displayed on the terminal depends on the size of the terminal screen. If the terminal allows more than one size, the screen will be set to the smallest size which can contain the panel.

The panel definition file can be created using any NOS text editor. The NOS 2 Full Screen Editor is particularly well suited to this purpose because it allows you to create and display the panel image in screen mode, much as the finished panel will appear when displayed on the screen.

NOTE

To ensure the correct functioning of PDU and screen formatting facilities, files should be created and edited as NOS 6/12-bit display code files.



```
TRYIN
{ VAR RSIDE1 T=REAL F=E R=(0. 999999999.)
      HELP= 'Enter positive integer or real value'
  VAR RSIDE2 T=REAL F=E R=(0. 999999999.)
      HELP= 'Enter positive integer or real value'
  VAR RSIDE3 T=REAL F=E R=(0. 999999999.)
      HELP= 'Enter positive integer or real value'
  KEY NORMAL=(NEXT)
  KEY ABNORMAL=(F6)}



                To find the area of a triangle:



               Enter values for Side A: ___________

                                Side B: ___________

                                Side C: ___________




                       Press:  NEXT to execute.
                               F6 to quit.
```

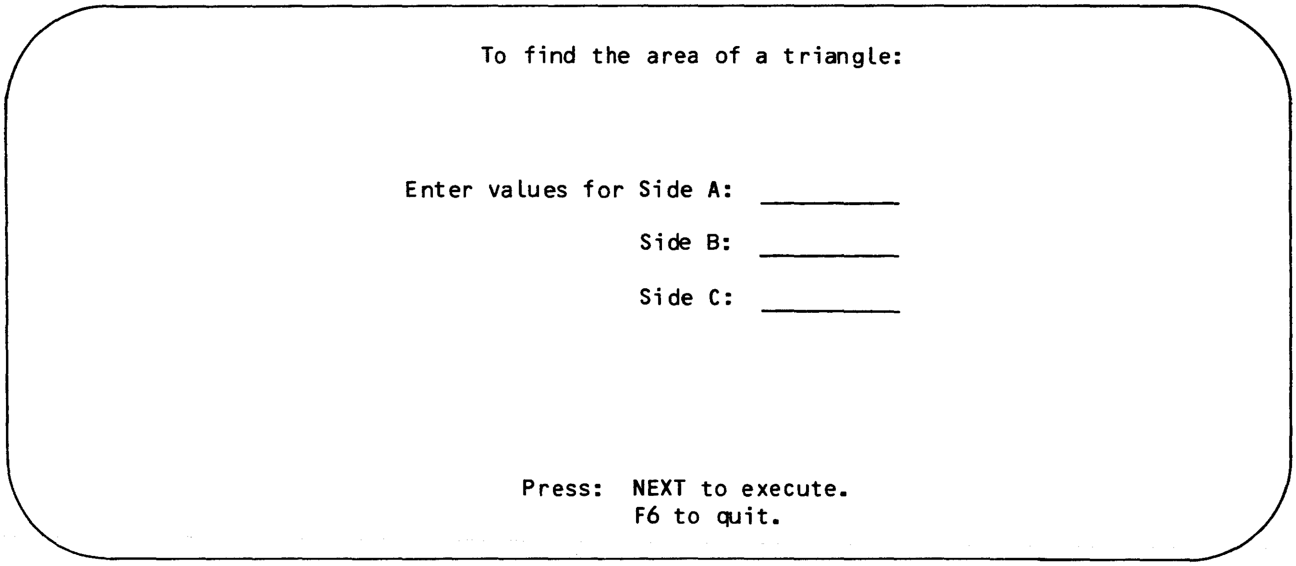
Figure 2-1. Panel Definition File

TITLE LINE

The title line is optional and is included to provide compatibility with NOS text record formats. If it is included in a panel definition file, the title line must be the first line in the file, must start in column one, and must contain only the name of the file, typed in uppercase characters.

DECLARATION SECTION

The declaration section defines the characteristics of any variable (input/output) fields in the panel and any nondefault terminal or display features, such as inverse video, blinking characters, or specially defined function keys. The declaration section is composed of a number of declaration statements. Each declaration statement defines a unique variable field or display feature (or combination of features).



```
                    To find the area of a triangle:


          Enter values for Side A: _________

                           Side B: _________

                           Side C: _________




                  Press:  NEXT to execute.
                          F6 to quit.
```

Figure 2-2. TRYIN Panel Display

The beginning and end of the declaration section are marked by the opening and closing braces, respectively. The opening brace must be the first character in the first line of the definition file, or the first character in the first line following the title line, if a title line is used. Any characters following the closing brace on the last line of the declaration section are ignored.

Format of Declaration Statements

Each declaration statement consists of a statement name followed by one or more parameters. Statement names and parameters are separated by at least one blank space. Declaration statements are written in free format; multiple statements, separated by semicolons, can be written on the same line and a single statement can be continued on multiple lines by ending each continued line with an ellipsis.

Declaration statement parameters are of the form keyword=value. If parameters are specified in the order shown in the format descriptions, the keyword and equal sign may be omitted. All keywords can be abbreviated, using only the first character of the keyword name.

Declaration statements can be written in uppercase or lowercase; PDU does not distinguish between uppercase and lowercase characters except for character strings enclosed in apostrophes (´´). Comments are inserted into the declaration section by enclosing them in quotation marks (" "). PDU ignores all data enclosed in quotation marks.

The following example shows a sample declaration section that defines five variable fields. The first three lines define three variables called A, B, and C. These variables are defined as type character, type integer, and type real, respectively. The next line defines a fourth variable called PAGE, which is of type character. The declaration statement for the fifth variable, NUMBER, begins on the fourth line and continues onto the two following lines. NUMBER is an integer variable with an initial value of 0.

```
{var name=a type=char
 var n=b    t=int
 var c      real
 var page char; var number ...
                int...
                0}
```

Physical and Logical Attributes

Some of the declaration statements allow you to specify physical or logical display attributes. You can assign these attributes to particular character strings in your panel to highlight important information or distinguish between different types of data.

Physical attributes explicitly identify display characteristics you choose to use in various situations. Examples of physical attributes include blinking, alternate intensity, inverse video, and color.

When writing application programs using physical display attributes, remember that all of these attributes are not available on all terminals. If an attribute is not available, it may be mapped into another attribute or ignored. Refer to appendix G for more information on which attributes are available on the system-defined terminals.

Logical attributes specify display characteristics in terms of the logical function of a character string. The logical attributes recognized are:

- Input text
- Output
  - Text
  - Italic (alternate font)
  - Title
  - Informative message
  - Error message

For a particular terminal, each of these logical attributes can have a unique set of physical attributes associated with it. When you assign a logical attribute to a character string, you cause the user's terminal to display the character string using the associated physical display characteristics for that terminal.

There are a number of advantages to using logical, rather than physical, attribute specifications:

- Logical attributes allow you to specify that different types of data are to be displayed differently without explicitly defining the physical display characteristics for each type of data.
- Logical attributes provide flexibility with respect to differing terminal models and capabilities. Since all terminal-dependent display characteristics are handled in the terminal definition software, panels defined in terms of logical display attributes do not require modification for new or different terminal models.
- Logical attributes promote uniformity in panel formats.

Declaration Statements

Table 2-1 gives a brief description of each of the declaration statements. The table is followed by a detailed description of each statement and the maximum number of times you can use the statement in one file.

Table 2-1. Declaration Statements

Statement	Description
ATTR	Defines physical or logical display characteristics used in the panel (maximum of 32).
BOX	Defines the character that indicates positions of lines and boxes in the panel (maximum of 32 with up to 256 distinct edges, corners, or intersections).
KEY	Defines function keys recognized by the program (maximum of 30).
PANEL	Defines an overlay panel.
TABLE	Defines a variable table (maximum of 32).
TABLEND	Indicates the end of the list of variables associated with a TABLE statement (maximum of 32).
VAR	Defines a variable field (maximum of 256).

The statement descriptions use the following format conventions:

Convention	Description
_ (underline)	Underlined characters indicate acceptable abbreviations for parameter keywords and values. Keywords and the following equal sign can be omitted if parameters are specified in the order shown in the format specifications.
() (parentheses)	Parentheses indicate that more than one value can be specified for a parameter. Individual values in a list of values must be separated by at least one space.
[] (brackets)	Brackets indicate optional parameters. Parameters listed vertically within brackets indicate that only one of the listed parameters can be specified.

For clarity of presentation, parameters shown in the statement formats are listed on separate lines using ellipses. When writing declaration statements, however, you may use any of the format options described under Format of Declaration Statements earlier in this section.

ATTR Statement

The ATTR statement defines a set of delimiters and associates them with one or more displayable attributes. Character strings bracketed by the delimiters in the image section are displayed (in the panel) with the associated display attributes. An ATTR statement can specify either a logical attribute or one or more physical attributes, but logical and physical attributes cannot both be used in the same statement.

The format of the ATTR statement is:

```
ATTR DELIMITERS='xy'...
    [PHYSICAL=(attr1 attr2 ... attrn)]
    [LOGICAL=attr                    ]
```

The ATTR statement parameters are:

Parameter	Description
DELIMITERS='xy'	x specifies the beginning delimiter and y specifies the ending delimiter that will surround the fields or strings to have the attribute or attributes being defined. x and y can be the same or different characters. The delimiters must be enclosed in apostrophes.
PHYSICAL=($attr_1$ $attr_2$ ... $attr_n$)	Specifies a physical display attribute or combination of attributes to be associated with the delimiters. The PHYSICAL parameter cannot be specified if the LOGICAL parameter is specified. If more than one attribute is specified, the attribute list must be enclosed in parentheses. An attribute list can contain one or more of the following physical attributes:

Attribute	Description
ALTERNATE	Alternate intensity character display.
BLINK	Blinking character display.
INVERSE	Inverse video display.
UNDERLINE	Underlined character string.
BLACK RED GREEN BLUE YELLOW MAGENTA CYAN WHITE	Colors.

Parameter	Description
LOGICAL=attr	Specifies a logical display attribute to be associated with the delimiter. The LOGICAL parameter cannot be specified if the PHYSICAL parameter is specified. attr can be any of the following logical attributes:

Attribute	Description
INPUT	Input text.
TEXT	Output text.
ITALIC	Alternate output text.
TITLE	Titles.
MESSAGE	Informative message text.
ERROR	Error message text.

Example:

The following ATTR statement defines a combination of physical display attributes. These attributes define the display characteristics for any character strings delimited by brackets in the definition file image section.

```
ATTR '[]' P=(BLINK RED)
```

BOX Statement

The BOX statement defines a termination character for the panel. The termination character is used to define endpoints or corners of lines, rectangular boxes, and other line figures. More than one termination character can be defined for a single panel, but each must be defined in a separate BOX statement.

Some terminals have special line drawing capabilities that allow you to display figures constructed of horizontal and vertical lines. PDU allows you to use these capabilities to add boxes or other line drawings to your panels.

You draw figures in the panel image using three different characters. Vertical lines are represented by the vertical bar, which may appear as | or ¦ depending on the terminal. Horizontal lines are drawn with the dash (-). The last character is the termination character, which defines corners or endpoints of a line. You may use any character as the termination character, but you must first define the character using a BOX declaration statement in the declaration section. If you define the asterisk as the termination character, a horizontal line will look like this:

```
    *-------------------------------*
```

While a rectangular box looks like this:

```
    *-------------------------------*
    |                               |
    |                               |
    *-------------------------------*
```

Here are some important points to remember when creating line drawings in your panels:

- You may define more than one termination character for a panel. Since you can associate any of the physical or logical display attributes with a given termination character, using more than one termination character allows you to specify different display attributes for different figures.

- Different terminal models vary in their ability to display line drawings. Terminals capable of replacing your line drawing characters with neatly drawn lines will do so, but other terminals may only be able to reproduce the characters you have used in your panel image.

The format of the BOX statement is:

```
BOX TERMINATOR='c' ...
   [WEIGHT=weight ...
   [PHYSICAL=(attr1 attr2 ... attrn)]
   [LOGICAL=attr                    ]
```

The BOX statement parameters are:

Parameter	Description
TERMINATOR='c'	Defines the line termination character. c can be any printable graphic character and must be enclosed in apostrophes. You cannot mix different termination characters within the same connected line figure. For example, you must use the same termination character for all four corners of a rectangle.
WEIGHT=weight	Specifies the line weight for lines or figures defined by the termination character. Values that can be specified for weight are FINE, MEDIUM, and BOLD; FINE is the default value.
PHYSICAL=($attr_1$ $attr_2$ ... $attr_n$)	Specifies a physical display attribute or combination of attributes for lines drawn using this termination character. The PHYSICAL parameter cannot be specified if the LOGICAL parameter is specified. If more than one attribute is specified, the attribute list must be enclosed in parentheses. The physical attributes that can be specified are listed in the ATTR statement description.
LOGICAL=attr	Specifies a logical display attribute for lines drawn using this termination character. The LOGICAL parameter cannot be specified if the PHYSICAL parameter is specified. The logical attributes that can be specified are listed in the ATTR statement description.

KEY Statement

The KEY statement defines which function keys terminate user input to the panel, allow match advancing, or provide help information. You may specify a normal or abnormal return for keys defined in a KEY statement. A normal return means that data the user has entered is checked against the validation requirements specified in the associated VAR statement(s). If any variable fails to meet validation requirements, the calling subroutine prompts for a corrected entry before returning control to the program. Thus, a normal return will not allow program execution to resume until all user input meets validation requirements. On the other hand, pressing a function key defined with an abnormal return causes input to be returned to the program immediately with no validation checking.

You may also define a key as a match advancing type key. If pressed within an input field with match validation defined, the next value in the match list will be placed in the field (starting at the first value in the list and wrapping back to it after all values have been displayed).

Your program can detect which function key was pressed by calling the SFGETK object routine. (SFGETK is described in section 3.) SFGETK returns a value to your program indicating which key the user pressed. Your program can use that value to determine what to do next.

If you define a KEY statement or statements for a panel, all function keys except the HELP key and the keys you define in the KEY statements will act as tab keys.

If you do not specify any KEY statements for a panel, all function keys except STOP and HELP will cause a normal return. STOP causes an abnormal return.

The KEY statement may be used to define any key as a HELP key. The HELP key (or any other key defined as help) functions as follows:

- If the cursor is positioned in a variable field for which a help string is defined (by the VAR statement HELP parameter), pressing the HELP key displays the help string in the message field (top line of the panel).

- If the cursor is positioned in a variable field for which no help string is defined and if the HELP key has been defined in a KEY statement, pressing the HELP key returns control to the application program (normally or abnormally as specified in the KEY statement).

- If the cursor is positioned in a variable field for which no help string is defined and if the HELP key was not defined in a KEY statement, pressing the HELP key displays the following message:

  Please enter

NOTES

When defining function keys, remember that only F1 through F6 and the NEXT key may be defined on some user-defined terminals. If you define keys at all, you must provide at least one key defined as normal or abnormal for the purpose of exiting any application screen. (This must be done since, if any keys are defined, all the rest of the undefined keys act as tabs.) For compatibility with Control Data software, all application programs should recognize the NEXT key, or its equivalent, as a normal return.

For more information on function keys available on the system-defined terminals, refer to appendix G.

The format of the KEY statement is:

KEY NORMAL=(key$_1$ key$_2$ ... key$_n$)
ABNORMAL=(key$_1$ key$_2$ ... key$_n$)
MATCH=(key$_1$ key$_2$ ... key$_n$)
HELP=(key$_1$ key$_2$ ... key$_n$)

The KEY statement parameters are:

Parameter	Description
NORMAL=(key$_1$ key$_2$ ... key$_n$)	Specifies the function key or keys that cause a normal return to the application program. If more than one key name is specified, the list must be enclosed in parentheses. To specify a shifted programmable function key, insert the word SHIFT before the key name. Key names that can be specified include any of the programmable function keys (F1 through F16) and any of the following CDC standard function keys: NEXT HELP BACK STOP FWD BKW UP DOWN Refer to appendix G for more information on these keys.
ABNORMAL=(key$_1$ key$_2$ ... key$_n$)	Specifies the function key or keys that cause an abnormal return to the application program. Key names that can be specified are the same as for the NORMAL parameter.
MATCH=(key$_1$ key$_2$ ... key$_n$)	Defines one or more function keys which can be pressed to provide values for an input field. When positioned in an input field that has match validation, pressing the defined key fills the field with the first value contained in the match list from the VAR statement. Pressing it again fills the field with the next value consecutively. It wraps to the first value when all other values have been used.
HELP=(key$_1$ key$_2$ ... key$_n$)	Defines a key or keys to be used for obtaining HELP information.

Example:

The following VAR and KEY statements define key F1 such that when you are positioned in the COLOR input field, pushing F1 will fill the field with the value red. Each time F1 is pushed, the field is filled with the next value in the string.

```
VAR COLOR MATCH=(red,green,blue,yellow)
KEY NORMAL=(FWD NEXT) MATCH=F1
```

Example:

The following KEY statement defines three function keys that cause a normal return and two keys with an abnormal return.

```
    KEY N=(NEXT HELP F1) A=(F6 STOP)
```

PANEL Statement

The PANEL statement identifies a panel as being either a primary panel or an overlay panel.

An SFSREA or SFSSHO subroutine call to a primary panel causes the screen to be cleared before the panel is displayed. An overlay panel modifies the current screen display without first clearing the screen. When an overlay panel is displayed, nonblank lines in the overlay panel overwrite the corresponding lines in the current screen display. Blank lines in the overlay panel leave the corresponding lines in the screen display unchanged.

Any number of overlay panels can be written to the screen simultaneously. Overlay panels may overwrite portions of other overlay panels.

Overlay panels may contain input and output fields, but all input variables appearing on the screen at any given time must belong to the same panel. In other words, if an overlay panel contains input variable fields, the panel must overwrite all displayed lines containing input variable fields.

The format of the PANEL statement is:

```
    PANEL NAME=panelname ...
          TYPE=type
```

The PANEL statement parameters are:

Parameter	Description
NAME=panelname	Specifies the name of the panel to be modified; if specified, it must be the same as the panel definition file name. This parameter is optional.
TYPE=type	Specifies the panel type as either PRIMARY or OVERLAY; PRIMARY is the default value. Currently, if PRIMARY is specified, the PANEL statement serves only to document the panel type. If type is specified, the panel is an overlay panel.

TABLE Statement

The TABLE statement, in conjunction with the VAR and TABLEND statements, defines a table data structure (two-dimensional array) for panel variables. Tables provide an easy way of manipulating repeated sets of variables. Each row of the table comprises one set of variables, so any variable value in the table can be accessed by using its variable name and row number. Rows are numbered consecutively, starting with row 1.

The format of the TABLE statement is:

```
    TABLE NAME=tablename ...
          ROWS=number
```

The TABLE statement parameters are:

Parameter	Description
NAME=tablename	Specifies the name of the table; the name can be from one to seven alphanumeric characters.
ROWS=number	Specifies the number of table rows; number must be an integer. The maximum table length is determined by the user´s terminal screen size. The results are unpredictable if the length of a defined table exceeds the number of text lines available on a terminal screen.

The actual table definition (as it appears in the declaration section) begins with a TABLE statement and ends with a TABLEND statement. The TABLE statement specifies the table name and the number of rows in the table. The TABLE statement is followed by a series of VAR statements, one for each variable in a table row. The TABLEND statement marks the end of the list of VAR statements associated with the table.

The following example shows a simple table definition as it might appear in the declaration section of a panel definition file:

```
    TABLE MAILIST 4
        VAR NAME
        VAR ADDR
        VAR PHONE
    TABLEND
```

This table definition defines a table called MAILIST, which consists of four rows of three variables each. The MAILIST definition implies a 4 by 3 variable array, which can be pictured like this:

	NAME	ADDR	PHONE
Row 1	name,1	addr,1	phone,1
Row 2	name,2	addr,2	phone,2
Row 3	name,3	addr,3	phone,3
Row 4	name,4	addr,4	phone,4

For each table variable defined in the declaration section, you must define a corresponding variable field in the image section. In other words, if you define a table with m variables and n rows, you must define m times n variable fields. As an example, the following lines could be used to define the variable fields for the MAILIST table:

```
          Name                          Address                       Phone

___________________   ______________________________   (612) _______
___________________   ______________________________   (612) _______
___________________   ______________________________   (612) _______
___________________   ______________________________   (612) _______
```

You can place the variable fields for a given table row on two or more image lines (that is, you do not have to put them all on the same line). The following is an alternate way of displaying the MAILIST table:

```
Name:     ________________________________________
Address:  ________________________________________   Phone: (612) _______

Name:     ________________________________________
Address:  ________________________________________   Phone: (612) _______

Name:     ________________________________________
Address:  ________________________________________   Phone: (612) _______

Name:     ________________________________________
Address:  ________________________________________   Phone: (612) _______
```

You can also put more than one table row on the same image line. For example, here is a third possibility for displaying the MAILIST table:

```
    Name          Address            Phone         Name          Address            Phone

__________   ____________   (612) _______   __________   ____________   (612) _______
__________   ____________   (612) _______   __________   ____________   (612) _______
```

When designing panels with tables, you can freely intermix constant data in the image section (such as the area codes in the above examples) with the table fields. Lines and boxes can be drawn between and around table variable fields.

TABLEND Statement

The TABLEND statement indicates the end of the list of VAR statements associated with the preceding TABLE statement.

The format of the TABLEND statement is:

```
TABLEND
```

VAR Statement

The VAR statement defines the characteristics of a panel variable field. Each VAR statement in the declaration section must have a corresponding variable field in the image section. VAR statements are associated with their corresponding variable fields by order of appearance: the first VAR statement defines the first variable field, the second statement defines the second variable field, and so on.

The format of the VAR statement is:

VAR NAME=fieldname ...
[TYPE=type]...
[VALUE=string]...
[FORMAT=c]...
[MATCH=(string$_1$ string$_2$ ... string$_n$)]...
[RANGE=(low high)]...
[PHYSICAL=(attr$_1$ attr$_2$ ... attr$_n$)
LOGICAL=attr]...
[ENTRY=condition]...
[IO=status]...
[HELP=string]

The VAR statement parameters are:

Parameter — Description

NAME=fieldname
Specifies a variable field name, one to seven characters long.

TYPE=type
Specifies whether the variable format is integer, character, or real. Values that can be specified for type are INT, CHAR, and REAL; CHAR is the default value.

VALUE=string
Specifies an initial value for the variable field. This value is displayed only when a panel is initially displayed by an SFSREA routine; that is, when a panel is opened by an SFOPEN subroutine call and read by an SFSREA call, with no intervening SFSWRI subroutine call. SFOPEN, SFSREA, and SFSWRI are described in section 3. The user can accept the displayed value or write over it. The value specified for string must match the variable type declared in the TYPE parameter, as follows:

Type — Description

CHAR
string must be a character string enclosed in apostrophes.

INT
string must be an integer in the N format (refer to the FORMAT parameter description).

REAL
string must be a real number in the E format (refer to the FORMAT parameter description).

Parameter — Description

FORMAT=c

Specifies the acceptable input format for the variable. This parameter does not reformat or otherwise affect the contents of the field. c can be any of the following format codes; however, the code specified must be compatible with the variable type as specified in the TYPE parameter. All formats allow trailing spaces in the variable field unless (MUST FILL) is specified for the ENTRY parameter.

Code — Description

X — Allow any characters; this is the default value if TYPE=CHAR is specified.

A — Allow only alphabetic characters.

9 — Allow only numeric characters.

N — Allow numeric characters with or without a leading sign; this is the default value if TYPE=INT is specified.

$ — Allow currency characters. A leading $ character is ignored and up to two digits are allowed after the decimal point. Commas are ignored.†

YMD — Allow date entry in YY/MM/DD format.

MDY — Allow date entry in MM/DD/YY format.

DMY — Allow date entry in DD/MM/YY format.

E — Allow real number entry in a format corresponding to the FORTRAN E format; that is, a leading sign, decimal point, and signed exponent (scientific notation) are allowed in addition to the digits that comprise the base of the number. This is the default value if TYPE=REAL is specified.

The format codes compatible with each variable type are as follows:

Type	Compatible Codes
CHAR	Any
INT	9, N, $, Y, M, or D
REAL	9, N, $, Y, M, D, or E

†If your site has so chosen, the meaning of the comma and decimal point may be reversed. That is, the comma may serve as the radix indicator and the period as the digit separator symbol.

Parameter — Description

MATCH=(string$_1$ string$_2$ ... string$_n$)

Specifies a list of acceptable values the user can enter for the variable. This parameter is valid only for character type variables. The user can enter truncated forms of a string if enough characters are entered to uniquely identify the string. If a string contains nonalphanumeric characters, you must enclose it in apostrophes; otherwise, apostrophes are optional.

RANGE=(low high)

Specifies a range of acceptable values for type integer or type real variables. low is the lower limit and high is the upper limit. Both low and high must be of the type specified for the variable.

For range validation purposes, integer variables with a FORMAT=$ specification are implicitly scaled (multiplied by 100). For example, an integer value of $1.50 falls within the range (125 200).

PHYSICAL=(attr$_1$ attr$_2$ ... attr$_n$)

Specifies a physical display attribute or combination of attributes to be associated with values displayed in the variable field. The PHYSICAL parameter cannot be specified if the LOGICAL parameter is specified. If more than one attribute is specified, the attribute list must be enclosed in parentheses. The physical attributes that can be specified are listed in the ATTR statement description.

LOGICAL=attr

Specifies a logical display attribute to be associated with values displayed in the variable field. The LOGICAL parameter cannot be specified if the PHYSICAL parameter is specified. The logical attributes that can be specified are listed in the ATTR statement description.

ENTRY=(condition)

Specifies special conditions pertaining to entry of the variable. Values that can be specified for condition are:

condition	Description
MUST ENTER	The user must enter a value for the variable.
MUST FILL	The user entry must fill the variable field; no trailing spaces are allowed.
UNKNOWN	The user may enter an asterisk when unsure of what to enter.

Parameter

Description

IO=status

Defines the input/output status of the variable field associated with this VAR statement. Values that can be specified for status are:

status	Description
(IN OUT)	The field is an input/output field; this is the default value.
OUT	The field is output-only; the program can display data in the field, but the user cannot enter data in the field.
IN	The field is input-only; data is never displayed in the field, either when entered by the user or during a program WRITE operation.

Some terminals do not support input-only fields. On these terminals, pressing any function key causes all input-only fields to be overwritten with spaces.

HELP=string

Defines a line of help text for the variable; string is a character string of up to 79 characters. The help string defined by this parameter appears in the message field (top line of screen, left-justified) under either of two conditions:

- The user presses the HELP key while the cursor is positioned in this variable field.
- Input to this field does not pass validation.

Validation of Variable Input Values

Calling either the SFSREA or SFSSHO object routine causes any user input to a panel to be read and validated. (SFSREA and SFSSHO are described in section 3.) Validation involves checking input values entered by the user against the validation requirements specified in the TYPE, FORMAT, MATCH, RANGE, and ENTRY parameters of the associated VAR statement. If all input values pass the validation checking, they are returned to the calling program, and program execution continues.

If one or more values fails validation, a message appears in the message field, and the screen cursor moves to the beginning of the variable field in error. The message field is left-justified in the top line of the panel. If you have defined any help text for the field in error (using the VAR statement HELP parameter), the help text is displayed in the message field. If no help text is defined for the field, the following default prompt appears in the message field:

```
Please correct
```

When the user enters a corrected value for the field and resubmits the panel input to the program, the entire process is repeated for the next variable field in error, if any.

On a normal return, execution of the calling program is not resumed until all erroneous input values are corrected. By defining a function key or keys that specify an abnormal return, however, you provide a way for the user to bypass validation checking. An abnormal return is a return in which the SFSREA or SFSSHO routine reads the input data and passes it to the calling program without performing validation checking. Both normal and abnormal returns are defined using the KEY declaration statement.

Any input erroneously entered outside an input field is blanked out by screen formatting. Normal input validation will then occur if the user has pressed a function key defined as a normal termination key. If there are no other input errors, the message

```
    Please confirm
```

is displayed to give the user an opportunity to verify that the information on the screen is correct.

IMAGE SECTION

The image section begins on the first line following the declaration section and continues to the end of the definition file. As the name implies, the image section contains an image of the panel showing how the panel is to appear on the screen. The image consists of any combination of: parameter or menu prompts that appear in the panel, other instructive or informative text, variable field markers, and characters representing lines or boxes drawn in the panel. All blank lines and spaces in the image section produce a like number of blank lines and spaces in the resulting panel.

You should usually leave the first line of a panel blank, since diagnostic messages generated by the screen formatting subroutines are displayed left-justified on the first line. If information is displayed on the first line, any diagnostic messages returned will overwrite the information on the screen.

When designing a panel, indicate the positions and lengths of variable fields by underlining the fields where you want them to appear in the image section. You may position the fields anywhere in the panel, since variable fields in the image section are associated with variable (VAR) declaration statements in the declaration section according to the order of appearance. The first VAR statement is associated with the first variable field, the second VAR statement is associated with the second variable field, and so on. The number of underlined characters in a variable field in the panel image should be the same as the length of the associated character variable declared in your application program.

The panel image you create in the panel definition file is the same as the resulting panel, with the following exceptions:

- Displayable attribute delimiters (as defined in the ATTR statement) are replaced by spaces, and the text between them is displayed with the attributes you declared.

- The underlines indicating variable fields in the definition file are not displayed in the panel. Instead, the variable fields are displayed using the input text display attributes defined for the terminal. For example, the Viking 721 displays input text with a solid underline, so a 5-character variable field that looks like this in the definition file:

  ```
  _ _ _ _ _
  ```

  looks like this when displayed in a panel on the Viking 721:

  ```
  _________
  ```

  When using terminals that do not support the underline attribute, you can identify the input fields by using delimiting characters which will appear on the panel. You may want to identify the input fields by writing your program to fill the field with a character such as an underscore. These characters would appear in the variable fields and would be typed over by the user.

- Image section characters defining lines or boxes are replaced by solid line drawings. (This action is subject to the capabilities of the user's terminal. A high-quality graphics terminal may be able to produce neat boxes and lines with all the attributes specified in the declaration section, while other terminals may only be able to reproduce the definition characters you used to define lines in the panel image. In the latter case, the image and the resulting panel will look very much alike.)

PDU COMMAND

The PDU command calls an interactive procedure that compiles a panel definition and stores the compiled panel in a user library. The compiled output is a load capsule which the procedure stores in a user library.

The user library to receive the load capsule must be a local file. If the library file you specify does not exist as a local file, PDU creates it. If you do not specify a library file, PDU uses a local file with the default name PANELIB, if one exists. If it does not exist, PDU creates a local file with the name PANELIB.

In the PDU command format, the parameter keywords and equal signs can be omitted if the parameters are specified in the order listed. The format of the PDU command is:

```
PDU,I=panel,L=listing,C=capsule,LIB=library
```

Parameter — Description

I=panel — Name of the panel definition file. The file must be a 6/12-bit display code, and the file name must be the same as the panel name. The I parameter has no default and must be specified.

L=listing — Name of the listing file. The listing file is a copy of the input file with error messages (if any) interspersed. The default listing file name is OUTPUT. If L=0 is specified, no listing is generated.

C=capsule — Name of the capsule file. The default capsule file name is CAPSULE. If C=0 is specified, the panel definition file is compiled and checked for compilation errors, but no capsule is generated.

LIB=library — Name of the library file to receive the encapsulated panel; must be a local file. The default library file name is PANELIB. If LIB=0 is specified, no library file is changed.

Since the PDU command is an interactive procedure, you can receive help information for the procedure and be prompted for parameter entries by entering:

```
PDU?
```

ULIB COMMAND

The ULIB command calls an interactive procedure used to create user libraries and add, modify, or delete individual records from a user library. Changes made to a user library or library record affect only the local copy of the library file; a modified library file can be made permanent by naming it in a REPLACE command. Because ULIB does not allow you to specify the type of record in a library (for example, CAP or PROC). All records in the library should have a unique name.

In the ULIB command format, the parameter keywords and equal signs can be omitted if the parameters are specified in the order listed. The format of the ULIB command is:

```
ULIB,OP=operation,REC=record,LIB=library
```

Parameter	Description
OP=operation	Specifies the library operation to be performed. The OP parameter must be specified. Values that can be specified for operation are:

operation	Description
C	Create a new user library.
A	Add a record to a user library.
D	Delete a record from a user library.
R	Replace a record in a user library.
F	Fetch a record from a user library and make it a local file. This operation does not modify the local library file.

Parameter	Description
REC=record	Name of the record to be added, deleted, replaced, fetched, or stored in a user library. The REC parameter must be specified.
LIB=library	Local file name of the library to be created or accessed. For any of the actions A, C, D, or R, ULIB returns the original file and creates a new local file; therefore, ULIB cannot modify a direct access permanent file. The LIB parameter must be specified.

Since the ULIB command is an interactive procedure, you can receive help information for the procedure and be prompted for parameter entries by entering:

```
ULIB?
```

SCREEN FORMATTING OBJECT ROUTINES 3

Panels used by application programs are defined using the PDU utility and are stored in libraries. The screen formatting object routines described in this section allow your FORTRAN5, COBOL5, or Pascal 1.1 program to retrieve panels from the libraries they are stored in and use them to perform terminal input and output operations. Some of the screen formatting object routines are directly involved in the entry or display of input and output data at the terminal. Others deal with related tasks, such as determining cursor positions.

NOS SYSTEM CONSIDERATIONS

When writing application programs that use screen formatting, you should be aware of some of the ways that the screen formatting object routines interact with NOS. This subsection describes these interactions in the areas of library usage and terminal status determination.

LINKING TO SCREEN FORMATTING ROUTINES

The screen formatting object routines are contained in a system library named SFLIB. A FORTRAN5, COBOL5, or Pascal 1.1 program using these routines must link up to them using the CYBER Loader.

The following NOS procedure contains commands to load, compile, and execute a FORTRAN program using screen formatting object routines. The source program in this example is called MYSOURC, and the absolute program is stored in a file called MYPROG.

```
.PROC,TRIPROG*I,
MYSOURC"SOURCE FILE"=(*F),
LISTING"LIST FILE"=(*F,*N=LISTING).
REWIND,*.
FTN5,I=MYSOURC,L=LISTING.
LDSET,LIB=SFLIB.
LOAD,LGO.
NOGO,MYPROG.
MYPROG.
REVERT,NOLIST.
EXIT.
REVERT,ABORT.TRIPROG
```

If the source program is written in COBOL, replace the line beginning with FTN5 with:

```
    COBOL5,I=MYSOURC,L=LISTING.
```

If the source program is written in Pascal, replace the line beginning with FTN5 with:

```
    PASCAL,I=MYSOURC,L=LISTING.
```

After the absolute program has been stored in file MYPROG, MYPROG can be saved in an existing user library for later use. The following NOS commands save MYPROG in a user library named MYLIB.

```
    GET,MYLIB.
    ULIB,R,MYPROG,MYLIB.
    REPLACE,MYLIB.
```

If MYLIB is a direct access permanent file, use:

```
    ATTACH,LIB=MYLIB.
    ULIB,R,MYPROG,LIB.
    ATTACH,MYLIB/M=W.
    REWIND,LIB.
    COPY,LIB,MYLIB.
```

To make MYPROG callable as a command, insert the following commands in your prologue if MYLIB is an indirect access file. If MYLIB is a direct access file, use ATTACH instead of GET.

```
    GET,MYLIB,PANELIB/UN=username.
    LIBRARY,MYLIB,PANELIB.
```

The LIBRARY command in this example establishes MYLIB (which contains MYPROG) and PANELIB as libraries within the global library set. Assuming that PANELIB contains the panels for MYPROG, MYPROG can now be called simply by entering the command:

```
    MYPROG
```

You may store the program and its panels in the same library. Refer to the NOS 2 Reference Set, Volumes 2 and 3, for further information on global libraries and prologues.

DISPLAYING YOUR PANEL

After you have compiled and stored your panel, you can display the panel by entering:

```
    SHOW,panelname.
```

This command calls an interactive procedure which displays the panel without your having to write a program to display it. panelname is the name of the compiled stored panel file in user library PANELIB or in a global library.

PANEL LIBRARY SEARCH ORDER

When a panel is referenced in a screen formatting object routine call, the object routine searches panel libraries in the following order:

- A local file named PANELIB
- A global library file
- The system library called PANELIB

SCREEN AND LINE MODES

The screen formatting object routines must know what terminal model is in current use. Before a program using screen mode displays can be run, either the application user or the procedure that executes the application program must enter a SCREEN or LINE command identifying the terminal.

The formats of these commands are:

```
LINE,model
   and
SCREEN,model
```

model is a user-defined (or site-defined) mnemonic which identifies a terminal. The mnemonic, which can be up to six characters in length, is the name of a compiled and stored terminal definition file. Entries for the seven system-defined terminals are:

Entry	Terminal
721	Viking 721
722	CDC 722
VT100	DEC VT100
Z19	Zenith Z19 or Heathkit H19
ADM3A	Lear Siegler ADM3A
ADM5	Lear Siegler ADM5
T4115	Tektronix 4115

(For more information on defining a terminal, refer to section 5.)

For example, either of the following commands informs the system that the user terminal is a Viking 721:

```
LINE,721

SCREEN,721
```

After the screen command has been entered, the screen formatting object routines, when called in an executing program, will set the terminal to screen mode and will have access to the terminal-dependent information required to perform data input and output functions.

PROGRAMMING CONSIDERATIONS

Panel-oriented input and output operations are easily integrated into application programs using the screen formatting object routines described in this section. Some considerations pertaining to panel usage in application programs follow.

CALL FORMATS

A FORTRAN5, COBOL5, or Pascal 1.1 application program calls the screen formatting object routines using the standard subroutine call format for the language being used.

A FORTRAN call to an object routine is formatted as follows.

CALL objrtn(p_1,p_2,p_3)

objrtn The 6-character name of the object routine.

p_1,p_2,p_3 The object routine parameters.

For COBOL, the object routine call is as follows (the variable values are the same as for the FORTRAN call).

ENTER objrtn USING p_1 p_2 p_3.

For Pascal, the object routine call is as follows (the variable values are the same as for the FORTRAN call).

```
objrtn (p1,p2,p3).
```

All screen formatting routines called from a Pascal program must be declared as FORTRAN-compatible external procedures. Any parameters which return a value to the calling Pascal application must be declared with the VAR keyword. Variables containing panel names can be declared as PACKED ARRAY[1..7] OF CHAR. Character strings containing variable data can similarly be defined as packed character arrays.

VARIABLE TYPES

The object routine descriptions in this section specify the variable type required for each object routine parameter. Table 3-1 relates the variable type notation (shown under Type) used in the object routine descriptions to the corresponding FORTRAN and COBOL variable types.

Table 3-1. Variable Type Notation

Type	FORTRAN	COBOL	Pascal
char	CHARACTER	01-level display item	CHAR
int	INTEGER	01-level COMP-1	INTEGER
real	REAL	01-level COMP-2	REAL

INPUT AND OUTPUT VARIABLES

Input and output data passed between the program and a panel are transferred as a concatenated character string. In other words, all panel variable values handled by the read and write object routines (SFSREA, SFSWRI, and SFSSHO) are considered to be of type character (FORTRAN type CHARACTER, COBOL 01-level display item, or Pascal type CHAR). The variable values are concatenated, in the order of their appearance in the panel, into a single variable string.

For example, assume that a panel has three 5-character variable fields specifying types character, integer, and real, in that order. Also assume that a user enters the following values into these fields: CAT, 123, and 98.6. The resulting character string returned to the program is:

C	A	T			1	2	3			9	8	.	6	

Your program must convert the concatenated string into individual variable strings of the appropriate type. This conversion can be accomplished using the character manipulation and type conversion facilities of the programming language.

In FORTRAN for example, type conversion can be accomplished by reading and writing internal files. The following sequence of FORTRAN statements converts the character string from the preceding example into individual character, integer, and real variables (the variable string is read from a panel called SAMPLE):

```
      INTEGER I
      REAL R
      CHARACTER C*5, S*15
         .
         .
         .
      CALL SFSREA ('SAMPLE',S)
      READ(S,1)C,I,R
    1 FORMAT(A5,I5,F5.0)
```

NOS screen formatting also provides two object routines (SFGETI and SFGETR) that extract individual values from the concatenated string and convert them to integer or real variables, as required.

OBJECT ROUTINES

This subsection describes the screen formatting object routines listed in table 3-2. For each routine, the six-character object routine name is followed by a list of parameters enclosed in parentheses. This format is for presentation purposes only. Refer to Call Formats in this section for a description of the language-dependent subroutine call formats.

Table 3-2. Screen Formatting Object Routines

Object Routine	Description
SFCLOS	Unloads a panel after use by the application program.
SFCSET	Specifies the code set that the application program uses for input and output data.
SFGETI	Returns the integer value of a single variable field.
SFGETR	Returns the real value of a single variable field.
SFGETK	Determines the last function key pressed.
SFGETP	Determines the cursor position when a function key was pressed.
SFOPEN	Loads a panel and prepares it for use.
SFPOSR	Establishes a current row in a named table (used only with SFGETI and SFGETR).
SFSETP	Sets the cursor to a selected screen position.
SFSREA	Displays a panel and permits entry of variable values.
SFSSHO	Displays a panel with current variable values and permits entry or modification of variable values.
SFSWRI	Displays a panel with current variable values.

SFCLOS (panelname,mode)

The SFCLOS object routine closes (unloads) a panel. Once closed, a panel can no longer be accessed unless it is reopened by another SFOPEN object routine call. Unloading a dynamically loaded panel frees the central memory used by the panel. It is not necessary to close a panel before another panel can be opened. By default, the maximum number of panels that can be open at one time is 10. Refer to appendix E for information on how to change the default limit.

The mode parameter specifies whether or not the screen is cleared and the terminal reverts to line mode when the panel is closed. If the panel specified in an SFCLOS subroutine call is the last panel displayed by the program, the subroutine call should specify reversion to line mode.

While debugging a program, it may also be convenient to revert to line mode at other points within the program. Reverting to line mode clears the screen and allows the terminal to display messages describing compilation or execution errors that may have occurred.

The SFCLOS parameters are:

Parameter	Type	Description
panelname	char	The name of a previously opened panel.
mode	int	An integer value indicating whether or not the terminal reverts to line mode after the panel is closed. The mode parameter must be specified. Values that can be specified for mode are:

mode	Description
0	Screen mode; leaves the screen unchanged and leaves the terminal in screen mode.
1	Line mode; clears the screen and returns the terminal to line mode.
2	Line mode; leaves the screen unchanged and returns the terminal to line mode.

Examples:

```
CALL SFCLOS ('MYPANEL',0)

ENTER SFCLOS USING "MYPANEL" SCREEN-MODE.

SFCLOS ('MYPANEL',0);
```

SFCSET (codeset)

The SFCSET object routine specifies the code set used by the application program in processing subsequent data. If no SFCSET object routine call is made, 6-bit display code is used.

The SFCSET parameter is:

Parameter	Type	Description
codeset	char	The code set required by the program. Values that can be specified for codeset are:

codeset	Description
DISPLAY	Specifies 6-bit display code.
ASCII	Specifies 6/12-bit display code.
ASCII8	Specifies 7-bit ASCII code, right-justified in a 12-bit byte.

Appendix A provides a conversion chart showing the display code equivalents of ASCII and ASCII8 characters.

Examples:

```
CALL SFCSET ('ASCII8')

ENTER SFCSET USING "ASCII8".

SFCSET ('ASCII8 ');
```

NOTE

When using Pascal, the parameters must be exactly seven characters long (padded with spaces as needed).

SFGETI (fieldname,value)

The SFGETI object routine returns the current value of the named variable field as an integer value.

The SFGETI parameters are:

Parameter	Type	Description
fieldname	char	The field name of the variable as specified in the panel VAR statement.
value	int	The variable to which SFGETI will return the integer value of the field specified in fieldname (FORTRAN type INTEGER, COBOL COMP-1, or Pascal type INTEGER). A value of 0 is returned if the specified field is all blanks or if an invalid character was entered in the field. The value returned is influenced by the VAR statement FORMAT parameter as follows:

FORMAT Parameter	Value Returned
9 or N	An integer value.
X	An integer value, if any.
$	The value of the field multiplied by 100. For example, 2 is returned as 200, 2.50 is returned as 250, and so on.
YMD, or MDY, or DMY	The integer value of the data in YMD format. For example, the following format and entry combinations all return the value 830131:

Format	Field Entry
YMD	83/1/31
MDY	1/31/83
DMY	31/1/83

FORMAT Parameter	Value Returned
E	The truncated integer value. For example, a value of 2.5 is returned as 2, and .25 is returned as 0.

Examples:

```
CALL SFGETI ('FIELD1',I)

ENTER SFGETI USING "FIELD1" FIELD1.

SFGETI ('FIELD1 ',I);
```

SFGETR (fieldname,value)

The SFGETR object routine returns the current value of the named variable field as a real variable.

The SFGETR parameters are:

Parameter	Type	Description
fieldname	char	The field name of the variable as specified in the panel VAR statement.
value	real	The variable to which SFGETR will return the real value of the field specified in fieldname (FORTRAN type REAL, COBOL COMP-2, or Pascal type REAL). A value of 0 is returned if the field is all blanks or if an invalid character was entered in the field.

Examples:

```
CALL SFGETR ('FIELD2',R)

ENTER SFGETR USING "FIELD2" FIELD2.

SFGETR ('FIELD2 ',R);
```

SFGETK (type,value)

The SFGETK object routine returns values that define the last function key pressed.

The SFGETK parameters are:

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>type</td><td>int</td><td>The variable to which SFGETK will return an integer indicating whether the last function key pressed was a CDC standard function key or a programmable function key. The options for type are:
<table>
<tr><th>type</th><th>Description</th></tr>
<tr><td>0</td><td>Programmable function key.</td></tr>
<tr><td>1</td><td>CDC standard function key.</td></tr>
</table></td></tr>
<tr><td>value</td><td>int</td><td>The variable to which SFGETK will return an integer indicating the last function key pressed. For programmable function keys, the value corresponds to the keycap numbering (that is, the value for F1 is 1, for F2 is 2, and so on). A negative value indicates a shifted function key. For CDC standard functions, the values are:
<table>
<tr><th>value</th><th>Key</th></tr>
<tr><td>1</td><td>NEXT</td></tr>
<tr><td>2</td><td>BACK</td></tr>
<tr><td>3</td><td>HELP</td></tr>
<tr><td>4</td><td>STOP</td></tr>
<tr><td>5</td><td>DOWN</td></tr>
<tr><td>6</td><td>UP</td></tr>
<tr><td>7</td><td>FWD</td></tr>
<tr><td>8</td><td>BKW</td></tr>
</table></td></tr>
</table>

SFGETP (fieldname,index,row)

The SFGETP object routine returns values that define the last position of the screen cursor.

The SFGETP parameters are:

Parameter	Type	Description
fieldname	char	The variable to which SFGETP will return a value indicating the field name of the variable field in which the cursor was last positioned.
index	int	The variable to which SFGETP will return a value indicating the character position within the variable field where the cursor was last positioned. An index of 1 indicates the first position, an index of 2 indicates the second position, and so on.
row	int	The variable to which SFGETP will return a value indicating the row number of the variable field if the variable is an element of a table. If the variable is not part of a table, row is returned as 0.

Examples:

```
CALL SFGETP (CNAME,INDEX,IROW)

ENTER SFGETP USING DISPLAY-NAME COMP-1-INDEX COMP-1-ROW.

SFGETP (CNAME, INDEX, ROW);
```

SFOPEN (panelname,status)

The SFOPEN object routine loads a panel and prepares it for use. It also sets the terminal to screen mode if it is not already in screen mode. To locate the specified panel, the system searches first a library contained in a local file named PANELIB (if one exists) then the user´s global library set, and finally, the system libraries. SFOPEN does not display the panel on the screen.

A panel must be opened using SFOPEN before it can be used by any other object routine. If another object routine attempts to use a panel before the panel is opened, the program is terminated abnormally.

The SFOPEN parameters are:

Parameter	Type	Description
panelname	char	The name of the panel to be opened.
status	int	The variable to which SFOPEN will return a value indicating the results of the attempt to open a panel. A value other than 0 indicates that the panel could not be opened. Possible values for status are:

status	Significance
0	The panel was successfully opened.
1	The panel could not be found.
2	The panel capsule was incorrectly formatted, probably due to panel definition errors.
3	Too many panels are already open. By default, up to 10 panels can be opened at once. Refer to appendix E for more information.
4	The specified panel is already open.
5	Internal errors occurred; the dayfile contains an informative message. This return is provided so the application can attempt a recovery and exit.
6	No SCREEN or LINE command identifying the terminal has been entered.
7	The terminal in use is not supported by NOS screen formatting.

Examples:

```
CALL SFOPEN (´MYPANEL´,ISTAT)

ENTER SFOPEN USING "MYPANEL" COMP-1-STATUS.

SFOPEN (´MYPANEL´,STATUS);
```

SFPOSR (tablename,row)

The SFPOSR object routine establishes a current row in the named table and is used in conjunction with the SFGETI and SFGETR object routines. Before calling an SFGETI or SFGETR object routine that references a table variable, your program must call an SFPOSR object routine to specify the row number of the desired variable value. The row number established by an SFPOSR subroutine call remains in effect for all subsequent SFGETI and SFGETR object routines until it is changed by another call to SFPOSR.

The SFPOSR parameters are:

Parameter	Type	Description
tablename	char	The 1- to 7-character name of a table defined by a TABLE statement in a currently active panel.
row	int	The row number of a row in the named table. The value specified is an integer in the range of 1 to the maximum number of rows defined for the table.

Examples:

```
CALL SFPOSR ('TABVAR1',2)

ENTER SFPOSR USING "TABVAR1" COMP-1-ROW.

SFPOSR ('TABVAR1',2);
```

SFSETP (fieldname,index,row)

The SFSETP object routine sets the screen cursor to a selected input variable field in the displayed panel. SFSETP can be called prior to an SFSREA or SFSSHO subroutine call to modify the default variable entry sequence. The default sequence proceeds sequentially from the first variable field in the panel to the last.

The SFSETP parameters are:

Parameter	Type	Description
fieldname	char	The name of the variable field in which the cursor is to be positioned.
index	int	The character position within the variable field where the cursor is to be positioned. An index of 1 indicates the first position, an index of 2 indicates the second position, and so on.
row	int	The row number of the variable if the variable is an element of a table. A value of 1 indicates the first row, a value of 2 indicates the second row, and so on. If the variable is not part of a table, specify 0 for row.

Examples:

```
CALL SFSETP ('PLAINV',1,2)

ENTER SFSETP USING "PLAINV" ONE TWO.

SFSETP ('PLAINV ',1,2);
```

SFSREA (panelname,instring)

The SFSREA object routine permits the user to enter input data at the terminal. Data entered is returned to the application program in instring. If the panel has not been previously displayed on the screen, SFSREA clears the screen and displays the panel using initial variable values specified for the panel (specified by the VAR statement VALUE parameter). If the panel is an overlay, only those lines that the overlay will write are cleared from the screen by SFSREA.

The SFSREA parameters are:

Parameter	Type	Description
panelname	char	The name of the panel to be used for input.
instring	char	The variable to which SFSREA will return the input data entered at the terminal for the panel specified in panelname. The value returned is a single character string (FORTRAN type CHARACTER, COBOL 01-level display item, or Pascal type CHAR) formed by concatenating the contents of all variable fields in the panel. (For more information, refer to Input and Output Variables in this section.)

Examples:

```
CALL SFSREA ('MYPANEL',INSTR)

ENTER SFSREA USING "MYPANEL" IN-STRING.

SFSREA ('MYPANEL',INSTR);
```

SFSSHO (panelname,outstring,instring)

The SFSSHO object routine displays a selected panel with current variable values, and allows the user to enter additions or modifications to the variable values which is returned in instring. If the panel is not already displayed on the screen, SFSSHO clears the screen and displays it using outstring for the variable field values. If the panel is an overlay, SFSSHO clears only those lines that the overlay will write. SFSSHO is equivalent to an SFSWRI object routine followed by SFSREA.

The SFSSHO parameters are:

Parameter	Type	Description
panelname	char	The name of a panel to be used for data input and output.
outstring	char	The variable containing the character data to be displayed at the terminal. outstring is a single character string (FORTRAN type CHARACTER, COBOL 01-level display item, or Pascal type CHAR) formed by concatenating the contents of all variable fields in the panel. (For more information, refer to Input and Output Variables in this section.)
instring	char	The variable to which SFSSHO will return the contents of all panel variable fields after modification by the user. Modifications made by the user are displayed in the panel as they are entered. instring is a single character string (FORTRAN type CHARACTER, COBOL 01-level display item, or Pascal type CHAR) formed by concatenating the contents of all variable fields in the panel. (For more information, refer to Input and Output Variables in this section.)

The same character variable or item can be used for both instring and outstring.

Examples:

```
CALL SFSSHO ('MYPANEL',OUTSTR,INSTR)

ENTER SFSSHO USING "MYPANEL" OUT-STRING IN-STRING.

SFSSHO ('MYPANEL',OUTSTR,INSTR);
```

SFSWRI (panelname,outstring)

The SFSWRI object routine displays the current variable field values. If the specified panel is not already displayed on the screen, SFSWRI clears the screen and displays the panel using outstring for the variable field values. If the specified panel is already displayed as a result of a previous SFSREA, SFSWRI, or SFSSHO object routine, only the variable field values are rewritten; all other screen data remains unchanged. If the panel is an overlay, only those lines that the overlay will write are cleared by SFSWRI.

The SFSWRI parameters are:

Parameter	Type	Description
panelname	char	The name of a panel to be written.
outstring	char	The variable containing the character data to be displayed at the terminal. outstring is a single character string (FORTRAN type CHARACTER, COBOL 01-level display item, or Pascal type CHAR) formed by concatenating the contents of all variable fields in the panel. (For more information, refer to Input and Output Variables in this section.)

Examples:

```
CALL SFSWRI ('MYPANEL',OUTSTR)

ENTER SFSWRI USING "MYPANEL" OUT-STRING.

SFSWRI ('MYPANEL',OUTSTR);
```

NOS PROCEDURES IN SCREEN MODE 4

NOS screen formatting allows a terminal user to enter NOS procedure parameters or menu selections in screen mode. The screen formats are predefined by the system and do not require special procedures; any of your existing interactive procedures can be used in screen mode without modification. Screen mode procedure entry does provide some additional features, however, which can increase the usability of your procedures. Becoming familiar with the screen mode display features will help you to write procedures that make the most effective use of full-screen display terminals.

NOS procedures allow you to place a sequence of operating system commands into a file and execute the file as you would a program. In effect, you create your own operating system commands to perform repetitive tasks, such as printing a file or loading and executing a program. NOS procedures can include parameters that affect how the procedure file is executed. Typical parameters would specify file names, processing options, and file dispositions. When executed interactively, NOS procedures can prompt the user for required parameter values and can display help information for the procedure and for individual parameters.

This section describes how procedures are executed in screen mode and tells you how to write procedures for screen mode display.

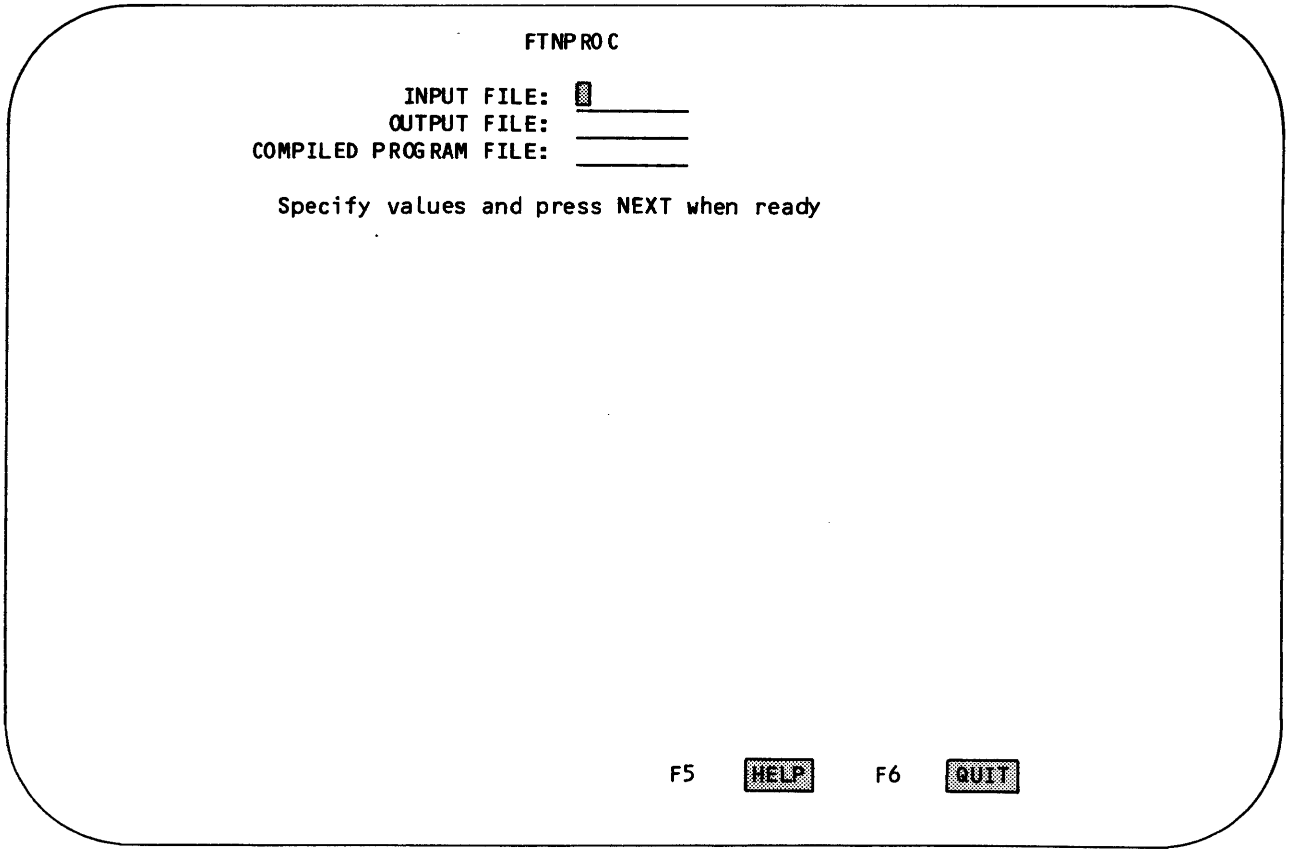
PROCEDURE EXECUTION

Screen mode display of NOS procedure parameters requires no special call format. When the user requests prompting for interactive procedure parameters, the parameters are displayed either in line mode or in screen mode, depending on the terminal status. If the user has entered a SCREEN command prior to the procedure call, the procedure parameters are displayed in screen mode; otherwise, the parameters are displayed in line mode.

When the user calls a procedure in screen mode, the terminal presents a screen display similar to that shown in figure 4-1 or figure 4-2. Figure 4-1 shows an interactive (*I format) procedure display, while figure 4-2 shows a menu (*M format) display.

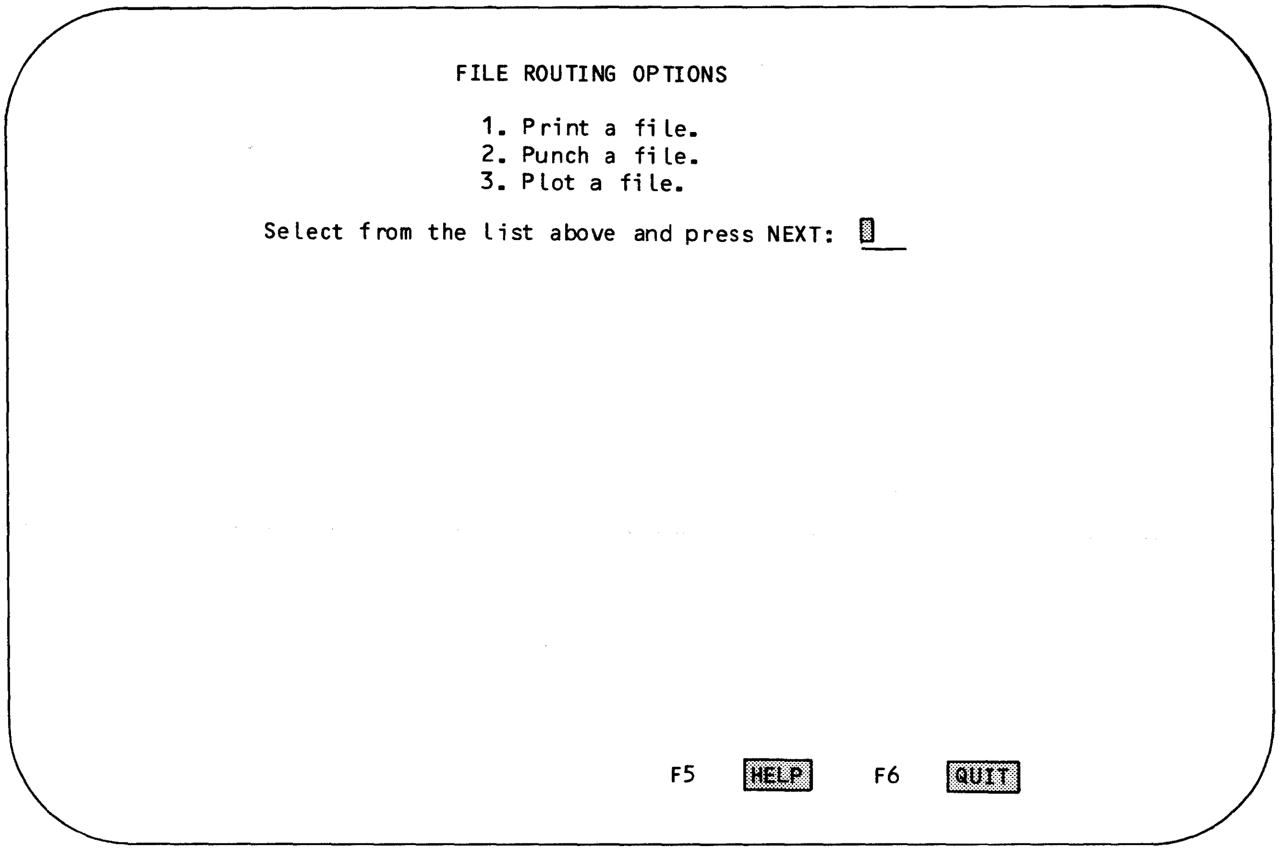
The parameter displays for a single procedure occupy up to nine screens of display text. The user can page forward and backward through the screen displays by pressing designated function keys. While paging through the parameter displays, the user can enter or modify parameter values in any order. To move from one parameter field to the next, the user presses the TAB key (the default entry sequence proceeds from the first field on the screen to the last). To enter parameters in nonsequential order or to modify values entered previously, the user moves the cursor to any parameter field on the screen using the cursor control keys.

When using any terminal that does not have protected fields, the TAB key must be followed by pressing the key corresponding to NEXT. On these terminals, you may press the TAB key more than once before pressing the NEXT key to position the cursor ahead more than one parameter field. Any programmable function key not defined in the panel definition file also functions as a logical tab.



```
                    FTNPROC

           INPUT FILE: ▯_______
          OUTPUT FILE: ________
COMPILED PROGRAM FILE: ________

  Specify values and press NEXT when ready
```

Figure 4-1. Interactive Procedure Display

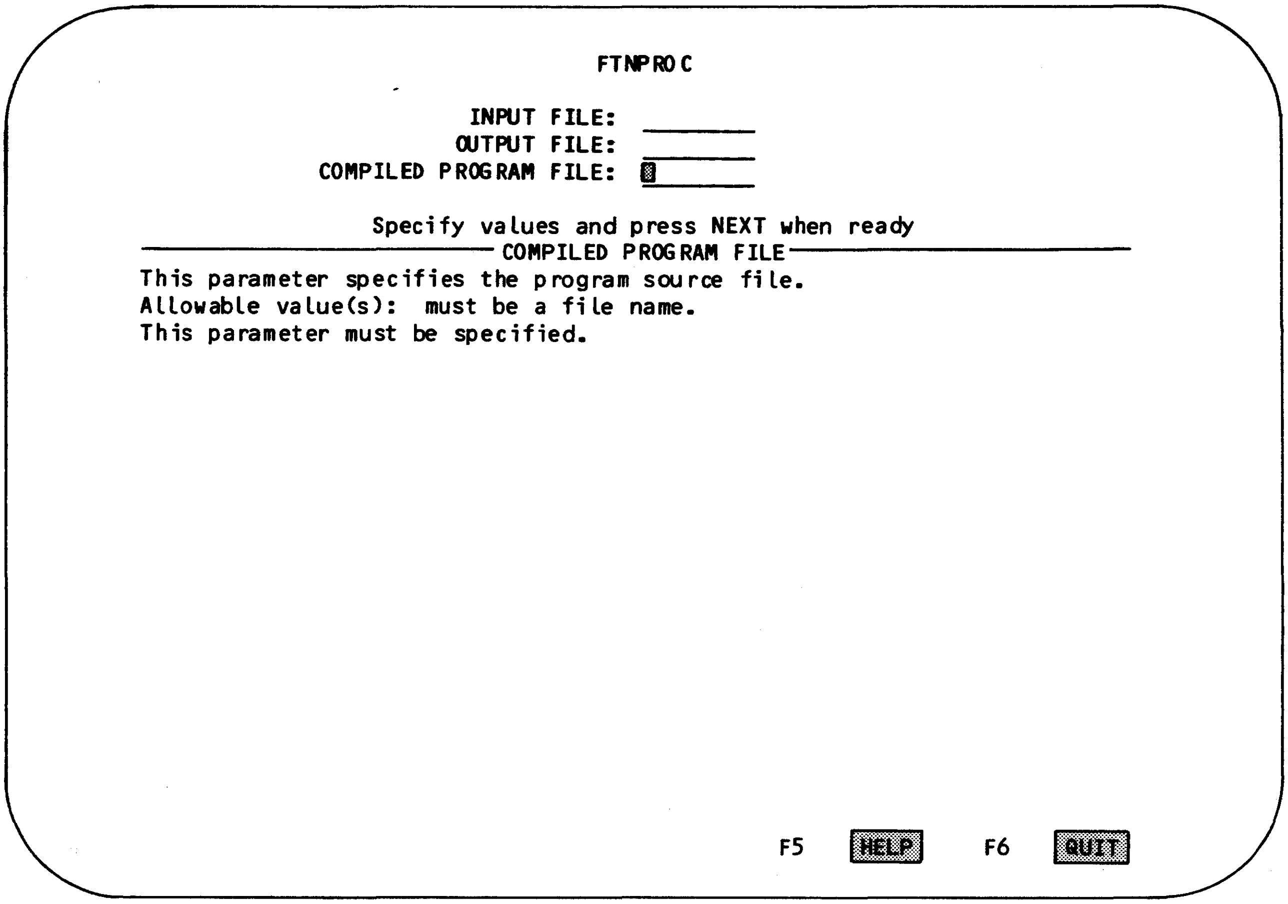


```
                    FILE ROUTING OPTIONS

                      1. Print a file.
                      2. Punch a file.
                      3. Plot a file.

Select from the list above and press NEXT: _
```

Figure 4-2. Menu Procedure Display

While paging through the displays, the user can also obtain help for the procedure or its parameters; a portion of the screen display is allocated for the help display. The function keys allow the user to page forward and backward through multiple pages of help text, if the help text does not fit on one screen. Figure 4-3 shows an example of a parameter display with help information.

After all required parameters have been entered, the user executes the procedure by pressing the NEXT key (carriage return). Parameter validation checks are performed in the same manner, regardless of whether the procedure is submitted in screen mode or line mode; if the user omits a required parameter or enters an incorrect value, the system prompts for a correct value before initiating execution of the procedure.

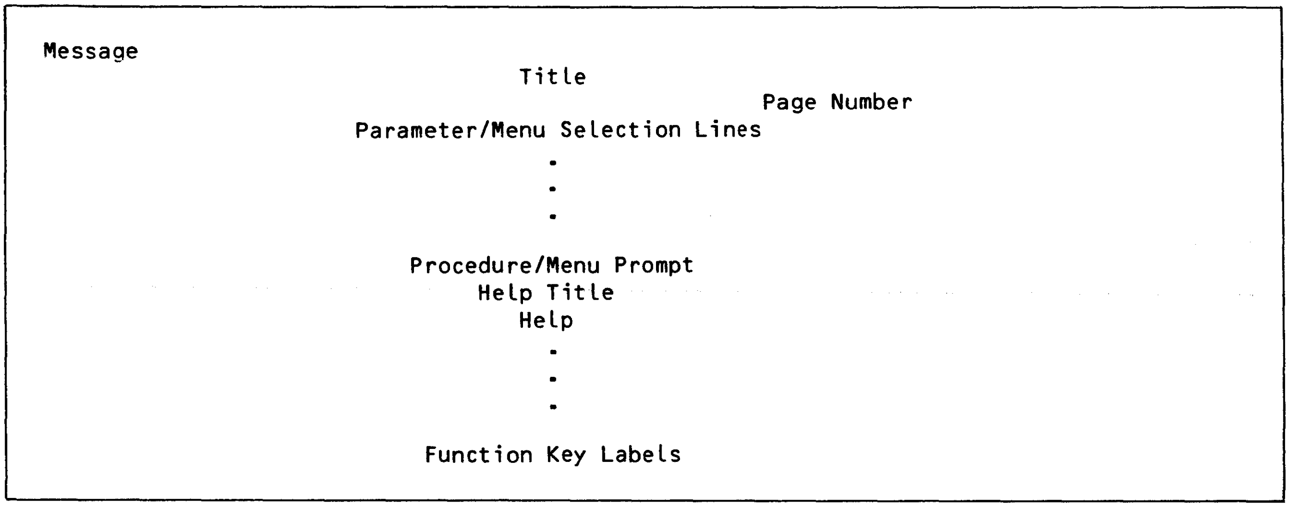


```
                              FTNPROC

                     INPUT FILE: ________
                    OUTPUT FILE: ________
           COMPILED PROGRAM FILE: ________

            Specify values and press NEXT when ready
--------------------- COMPILED PROGRAM FILE ---------------------
This parameter specifies the program source file.
Allowable value(s):  must be a file name.
This parameter must be specified.
```

Figure 4-3. Interactive Procedure Display with HELP Text

SCREEN MODE PROCEDURE FORMAT

Figure 4-4 illustrates the screen mode format used to display procedure parameters. The format contains six fixed-content lines. These lines are labeled Message, Title, Page Number, Procedure/Menu Prompt, Help Title, and Function Key Labels. The number of parameter/menu selection lines and help lines vary, depending on the terminal screen size and the number of lines required by the procedure. The minimum supported screen size is 16 lines of 80 columns.



```
Message
                                  Title
                                                     Page Number
                         Parameter/Menu Selection Lines
                                    .
                                    .
                                    .

                             Procedure/Menu Prompt
                                Help Title
                                  Help
                                    .
                                    .
                                    .

                             Function Key Labels
```

Figure 4-4. NOS Procedure Screen Format

The following paragraphs describe the components of the NOS procedure screen format as shown in figure 4-4.

MESSAGE

The message line informs the user when a parameter has been entered that does not meet the validation requirements specified in the procedure. The message consists of an output-only field of up to 79 characters, left-justified on the first line of the screen. When a message is displayed in the message line, the screen cursor is automatically placed at the beginning of the data field associated with the message.

The following message is displayed if the user fails to enter a value for a required parameter that does not have a defined help string.

```
Please enter
```

You can replace the phrase Please enter using the .ENTER directive. This directive is useful when writing procedures for non-English speaking users. The .ENTER directive format is:

```
.ENTER,string
```

string   Specifies a string of from 1 to 40 characters.

The following message is displayed when an invalid value has been entered:

```
PLEASE CORRECT value
```

value — Identifies the incorrect value entered. If value is longer than 64 characters, it is truncated to 61 characters, followed by an ellipsis, as shown in the following example:

```
PLEASE CORRECT this message is longer...
```

You can replace the phrase Please correct with another message using the .CORRECT directive. The .CORRECT directive format is:

```
.CORRECT,string
```

string — A string of from 1 to 40 characters.

The system returns only one error message at a time, even if the screen contains more than one error. When the user corrects an indicated error and resubmits the procedure (by pressing the NEXT key), the next error message, if any, appears. This process continues until all errors are corrected. The user may correct any number of errors before resubmitting a procedure.

TITLE

The title specified in the procedure header is displayed, centered, on the second line of the screen. If no title is specified in the procedure header, the procedure name is used as a default title.

PAGE NUMBER

The page number line displays the number of the current page of parameters or menu selections. If all parameters or selections fit on one page, the page number field is blank. The format of the page number field is:

```
Page n
```

n — The page number.

You can replace the word Page with another word or phrase using the .PAGE directive. The .PAGE directive format is:

```
.PAGE,string
```

string — A string of from 1 to 40 characters.

PARAMETER/MENU SELECTION LINES

The page number line is followed by a variable number of lines which prompt the user for parameter entries or menu selections. The number of parameter/menu selection lines available on each page depends on the terminal type but typically will range from 6 to 17 lines. If all parameters do not fit on one page and leave space for help text on the same page, the parameter descriptions are continued on one or more additional pages. Following are the prompt formats for interactive parameters and menu selections.

Interactive Parameter Prompts

A procedure parameter specification uses one of the following three prompt formats. The right-hand column shows the corresponding screen prompt generated by each specification format.

Parameter Prompt Format	Full-Screen Prompt
Parameter=	Parameter:
Parameter"Description"=	Parameter Description:
Parameter´Description´=	Description:

Regardless of which format is used, each parameter prompt is followed by a 1- to 40-character input field. The system indicates the length and position of the input field by underlining the field. Input characters are displayed in the field as the user enters them at the terminal.

Interactive parameter prompts are centered on the screen according to the length of the longest parameter description and input field length to be displayed.

The length of the input field for each parameter is that of the largest variable value that can be entered for the parameter. This length, in turn, is implied by the checklist pattern used in defining the parameter. The maximum variable lengths for each checklist pattern are as follows:

Checklist Pattern	Maximum Length
*F	Seven characters.
*A	Forty characters.
*K	A value equal to the length of the parameter name.
*Sn	A value equal to the maximum length (as specified by n) of the set.
literal string	A value equal to the number of characters in the literal string.

The following examples illustrate the formats that result from various interactive parameter specifications.

Parameter and Checklist	Prompt Generated
CSET=(A, D, A8)	CSET: _ _
I"- Input file"=(*F)	I - Input file: _ _ _ _ _ _ _
I´File to copy´=(*F)	File to copy: _ _ _ _ _ _ _
R´Rewind (Y or N)´=(Y, N)	Rewind (Y or N): _

Menu Selection Prompts

Menu selection prompts in both screen and line mode are preceded by a number, period, and space. The menu is centered on the screen according to the longest selection prompt in the menu. Prompts that are too long to fit on the screen are truncated on the right.

Procedure/Menu Prompt

The procedure/menu prompt line tells the terminal user what to do when he or she has finished entering parameters or menu selections. The prompt format for interactive procedures is:

```
Specify values and press NEXT when ready
```

Menu procedures prompt for a numeric value. The prompt format is:

```
Select from the list above and press NEXT: __
```

This prompt directs the user to select a menu item, enter the number of that item in the input field, and press the NEXT key.

You can replace either of the preceding prompts using the .PROMPT directive. The format of the .PROMPT directive is:

```
.PROMPT,string
```

string   A string of 1 to 40 characters.

HELP TITLE

The help title line appears on the screen only when help text is being displayed. The help title is centered in the line. It consists of the parameter or procedure name for which help is being displayed. To clearly separate help information from the parameter/menu selection information, a medium intensity horizontal line is drawn through the portions of the help title line not occupied by the title itself.

HELP

Help text appears in a variable number of lines that appear between the help title line and the function key labels. Six or more lines (depending on the terminal model) are available for help text displays. Help text can occupy more than the minimum number of help lines if the parameter prompts or menu selections do not require all lines that are available to them. The system displays as much of the help text as it can fit on the screen without overwriting parameter descriptions or menu selections.

There is no restriction on the length of help text you can write into a procedure. The terminal user can page forward or backward through the help text by pressing a function key. This feature is described in detail under Function Key Labels.

Two types of information are available to the terminal user through help texts: information on the procedure and its functions and descriptions of procedure parameters. You supply the help text for procedure and parameter information using the .HELP directive.

The terminal user obtains help by pressing the HELP key or by entering a question mark in a parameter field. To obtain help for a menu selection, the user enters the number of the selection followed by a question mark. For example, the entry 2? requests help information for menu selection 2. To remove help text from the screen, the user presses the BACK key.

FUNCTION KEY LABELS

The bottom line of the screen displays a series of descriptive labels, one for each active programmable function key. (The programmable function keys are labeled F1, F2, and so on.) Each label consists of a word or phrase describing the action of the associated key. For example, the key that requests help text (F5) is appropriately labeled HELP. The function key labels are displayed in inverse video, so they appear as a series of rectangular boxes across the bottom of the screen. Each box is preceded by the name of the key associated with the label.

Table 4-1 describes the function keys that are active for NOS procedure parameter displays.

Table 4-1. Programmable Function Keys

Key	Label	Description
F1	FWD	Displays the next page of procedure parameters or menu selections. If there is no next page, the F1 label does not appear.
F2	BKW	Displays the previous page of procedure parameters or menu selections. If there is no previous page, the F2 label does not appear.
F3	HELP FWD	Displays the next page of help text. If there is no next page, the F3 label does not appear.
F4	HELP BKW	Displays the previous page of help text. If there is no previous page, the F4 label does not appear.
F5	HELP	Displays help text as follows: ● Pressing the help key once displays parameter help for the parameter field at which the cursor is currently positioned. ● Pressing the help key a second time, without moving the cursor, displays help text for the procedure.
F6	QUIT	Terminates the procedure normally without executing the procedure.

You can replace the default function key labels using the .Fx directive. The .Fx directive format is:

```
.Fx,string
```

x Specifies an integer value from 1 to 6, corresponding to a function key from F1 to F6.

string Specifies a character string of from one to six characters.

The .Fx directive does not change the operation of the function keys. For example, F5 provides help, regardless of how it is labeled in the screen display.

On the Viking 721, some of the preceding operations can also be performed using the CDC standard function keys available on the Viking 721. The keys and their functions are as follows:

Key	Function
FWD	F1 (FWD)
BKW	F2 (BKW)
HELP	F5 (HELP)
STOP	F6 (QUIT)

Also, the BACK key can erase help text from the screen. This function may not be available on some terminals.

The .NOCLR directive inhibits the system from automatically clearing the terminal´s screen at the end of the procedure call (that is, once all required parameters are supplied). You can also specify a message to appear on the top line of the screen. Unless you specify a .NOCLR directive, the system clears the screen at the end of the call and sets the terminal to line mode, allowing any generated dayfile message to be displayed.

The .NOCLR directive is useful in procedures which call a program or a series of nested procedures. Using the .NOCLR directive in these situations prevents the screen from remaining blank for an undesirable length of time. The .NOCLR directive should not be used in unnested procedures or in the last (innermost) procedure in a series of nested procedures.

Format:

```
.NOCLR,message.
```

message Specifies a 1- to 40-character text string that appears on the screen. message can consist of both uppercase and lowercase characters.

TERMINAL DEFINITION UTILITY 5

Terminals using full-screen applications on NOS must be defined using the Terminal Definition Utility (TDU).[†] After compilation by TDU, the definitions are stored in libraries for use by the terminal support routines common to all full-screen products.

Any display terminal with certain minimal capabilities which can be defined using the TDU utility will work with any full-screen product.

TERMINAL CAPABILITIES

To be used with full-screen products, a terminal must have the following attributes:

- Uses asynchronous communications (as opposed to synchronous).
- Operates in character mode (as opposed to block mode).
- Has keys which move the cursor on the screen and transmit characters to the host computer so it can tell the cursor moved.
- Supports direct cursor positioning.
- Provides a clear screen operation.

The terminal should have the following additional attributes:

- A clear-to-end-of-line.
- A way to define at least six function keys.

[†] The system already has definitions of the following seven terminals. The terminal definitions used for these terminals are records on file TDUFILE under user name LIBRARY. You can modify these definitions to meet your particular needs. Across from the terminal model names in the following list are the corresponding records on file TDUFILE (on user name LIBRARY) that contain the terminal definitions.

Terminal	Terminal Definition
Viking 721	TDU721
CDC 722	TDU722
DEC VT100	TDUVT10
Zenith 719 or Heathkit H19	TDUZ19
Lear Siegler ADM3A	TDUADM3
Lear Siegler ADM5	TDUADM5
Tektronix 4115	TDUT415

In addition to those previously mentioned, the following terminal attributes are desirable:

- Eight to 32 function keys.
- Function keys should transmit a unique, identifying character sequence followed by a carriage return (CR) character.
- Host-definable tab stops (for use with the Full Screen Editor).
- Protected fields on the screen and tabbing between unprotected fields (for use with screen formatting). The tab key, like the cursor keys, must transmit characters to the host so it can tell the tab key was pressed.
- Line drawing graphic characters.

Other terminal features are supported by full-screen products, but those listed are heavily used. (The CR at the end of function key sequences provides added usability and is a feature of the Viking 721 terminal.)

TERMINAL DEFINITION FILE

Terminal keys are defined by typing definition statements into a text file and compiling the file using TDU. The text file must be in 6/12-bit display code.

Terminal definition statements are highly readable but can be tedious to type. A text file with all the statements already typed and formatted can be obtained by entering the command:

```
    GET,TDUIN/UN=LIBRARY
```

Edit this file and fill in the parameters to describe your terminal.

You will need your terminal hardware reference manual for filling in the file. TDUIN lists statements for all possible attributes and keys that can be supported by full-screen products. In the hardware reference manual there should be one or more tables listing the keys and attributes available on your terminal. After each key or attribute listed in these tables, the character sequence your terminal accepts or generates is listed. Use these character sequences to fill in the statement parameters in the TDUIN file. TDUIN contains directions (enclosed in quotation marks before each statement) which give more instructions on filling in the file´s directive parameters. Read these carefully. Not all attribute and key statements will apply to your terminal. Leave those which do not apply blank.

An example of a terminal definition file for the Viking 721 is shown at the end of this section.

The TDUIN file includes some statements for defining Full Screen Editor (FSE) keys. For more information on these statements consult the FSE User´s Guide.

NOTE

> If you use TDU to define any of your terminal keys, you must define your FSE keys either in your terminal definition file or your FSEPROC. The normal default FSE key definitions are no longer used once you define any terminal keys using TDU.

Compile your terminal definition file using the TDU utility and store it on TERMLIB. This load capsule will be used to define your terminal anytime you enter the SCREEN,model command with model being the MODEL_NAME you specified in your terminal definition file. To verify the creation or replacement of the capsule on your library file, get a catalog of the library and check for the terminal model name prefixed with a Z.

Before you start, it is wise to check whether someone else at your installation has already defined your terminal. Your installation probably makes a number of compiled definitions publicly available in the TERMLIB file on user name LIBRARY. To get a list of all the terminal models TERMLIB already has, enter the commands:

```
    GET,TERMLIB/UN=LIBRARY
    CATALOG,TERMLIB,R,U,N
```

STATEMENT FORMAT

The general format of a terminal definition statement is:

Statement_name keyword$_1$=value$_1$ keyword$_2$=value$_2$...
keyword$_n$=value$_n$

The statement_name and any of the keywords may be entered in either uppercase or lowercase. Keywords and equal signs may be omitted if values are entered in the order they are defined for the statement. The elipsis (...) is used to continue statements onto another line. More than one statement may be typed on the same line if the statements are separated by a semicolon.

Statement names may be entirely spelled out or may be abbreviated by using the first three characters of the first word and the first character of each following word. For example, the following are equivalent statement names:

function_key_leaves_mark
funklm

Keywords are usually abbreviated by the first character (but INOUT is abbreviated IO; IN is abbreviated I).

Comments may be used anywhere in a statement where blank spaces can appear (except within quotes). Comments are enclosed in quotes (") characters. Character strings are enclosed in apostrophes (´). For example,

"This is a comment."
´This is a character string´

The most frequently occurring parameter value in terminal definition statements is a list of characters. Lists must be enclosed in parentheses. These lists are obtained from the terminal hardware reference manual. Often the tables containing these character strings list more than one representation. Character values that you enter in the terminal definition file may be indicated in any one of the ways shown in the following example:

Value	Meaning
´A´	The ASCII character A.
101(8)	The character A as an octal number.
41(16)	The character A as a hexadecimal number.
65	The character A as a decimal number.
33(8)	The ASCII ESC character as an octal number.
ESC	The ASCII ESC character indicated by its standard designation. Standard designations of ASCII characters are shown in table A-1 in Appendix A.

For example, the following are valid terminal definition statements:

```
MODEL_NAME VALUE=´721´
BLINK_BEGIN OUT=(ESC 12(16) ´a´)
```

These examples show values as ASCII character strings (´721´,´a´), an ASCII character (ESC), and a hexadecimal number (12(16)).

If you are going to be using a character string more than once, you may want to define a variable name to have that value. This can be done by listing the variable name and its value at the beginning of the file before any of the TDU statements. The format is:

```
variable_name = (character string)
```

variable_name can be any string of alphanumeric characters and the underscore. It can be up to 256 characters in length. character string is the sequence listed in your terminal hardware reference manual for a particular attribute.

STATEMENT TYPES

Following is a list of the different types of statements. Details on the specific statements and their parameters are explained later in this section. TDUIN, the file to use for creating your terminal definition file, also lists the parameters and information for using them.

Statements — Description

Attribute — Describe general characteristics of the terminal. For example:

```
HOME_AT_TOP   VALUE=TRUE
```

Attribute statements have parameters appropriate for the characteristic being described. The VALUE parameter will usually be either TRUE or FALSE, or it may be some other alphanumeric value, depending on the terminal.

Cursor positioning — Describe the behavior of the cursor on the screen. The statement will have TYPE parameters describing the cursor movement. For example:

```
MOVE_PAST_SIDE      TYPE = WRAP_ADJACENT_NEXT
```

Screen size — Describe the size of the screen. For example:

```
SET_SIZE   ROWS=24 COLUMNS=80...
OUT=(re dc2 ´H´ rs dc2 ´!´)
```

This statement has the following parameters:

Parameter — Description

ROWS — The number of rows on the terminal.

COLUMNS — The number of columns on the terminal.

OUT — The sequence to be sent to the terminal. This sequence must be obtained from the terminal hardware reference manual.

Initialization Output — Describes terminal attributes set and cleared when the LINE or SCREEN command is executed. These statements may be repeated to allow entrance of long character strings for initializing the terminal.

Statements	Description
Input/output	Describe character sequences which can either be sent by the terminal or by the host computer. For example:

```
CURSOR_UP          INOUT=(VT)
```

Input/output statements have the following parameters:

Parameter	Description
INOUT=sequence	The character sequence transmitted to or from the host.
LABEL=string	A character string which identifies the corresponding keyboard key. For example: CURSOR_UP LABEL='CTRL-H' LABEL is optional.

Statements	Description
Input	Describe character sequences generated by the terminal keyboard and transmitted to the host computer. For example:

```
F1     LABEL = 'F1'    INPUT = (RS DC1 'h')
```

Input statements have the following parameters:

Parameter	Description
INPUT=sequence	The character sequence, not to exceed 256 characters, transmitted to the host. INPUT is required.
LABEL=string	A character string which labels the corresponding keyboard key. LABEL is optional.

Statements	Description
Output	Describe character sequences sent from the host computer to the terminal. For example:

```
BLINK_BEGIN     OUT=(12(16))
```

Output statements have the following parameter:

Parameter	Description
OUT=sequence	The character sequence, not to exceed 256 characters, transmitted to the terminal. OUT is required.

The categorization of statements as input, output, or input/output is based on what the full-screen products can actually do with a terminal. It might be, for example, that a terminal could generate a BLINK_BEGIN sequence from the keyboard, but programs, such as FSE, will not recognize such an input sequence, so BLINK_BEGIN is an output statement. Conversely, the terminal might not be able to recognize a sequence such as CURSOR_RIGHT if sent from the host, so it is acceptable to specify this as an IN parameter, even though CURSOR_RIGHT is an input/output statement. This tells the full-screen products to recognize CURSOR_RIGHT but not to try to send it.

REQUIRED CAPABILITIES

Some capabilities are required for the full-screen products to work correctly. These are:

CURSOR_HOME	
CURSOR_DOWN	
CURSOR_LEFT	
CURSOR_POS_BEGIN	(and possibly CURSOR_POS_SECOND AND_CURSOR_POS_THIRD, if these are used for your terminal)
CURSOR_POS_ENCODING	
CURSOR_RIGHT	
CURSOR_UP	
ERASE_PAGE_STAY OR ERASE_PAGE_HOME	
MODEL_NAME	
ERASE-END-OF-LINE	(not required but highly desirable)

There must also be a subset of the application function keys available and defined (a minimum of six). All statements that are required will be identified as such in their descriptions in the TDUIN file.

ATTRIBUTE STATEMENTS

The following statements may be used to describe terminal characteristics:

Statement	Parameter	Description
MODEL_NAME		The model name identifies the type of terminal being defined. The model name is used as the name of the definition in the TERMLIB file, and is the name used as the model name parameter on the SCREEN or LINE command. Required statement.
	VALUE=name	The model name may be a one to six alphanumeric character string. Lowercase letters are translated to uppercase.
COMMUNICATIONS		Identifies the type of communication the terminal uses. Required statement.
	TYPE=type	type refers to the terminal protocol. ASYNCH is the value used to indicate an asynchronous terminal.

Statement	Parameter	Description
CURSOR_POS_ENCODING		Tells how the cursor position output sequence is encoded. Most terminals fall in one of the categories below. Required statement.
	TYPE=encoding	Let a be the cursor_pos_begin, b the cursor pos_second, c the cursor_pos_third, x the horizontal position, and y the vertical position. The values for a,b,c,x,and y must be obtained from your terminal hardware reference manual. The general encoding format is: axbyc All terminals will have an a, x, and y at least. The value of encoding is interpreted as follows:

Encoding	Description
BINARY_CURSOR	The cursor positioning sequence is of the format: a (x+bias) (y+bias) or a (y+bias) (x+bias)
ANSI_CURSOR	X and y are generated as decimal graphic characters; for example, '12' rather than OC(16), with format:. a (x decimal) b (y decimal) or a (y decimal) b (x decimal) c
CDC721_CURSOR	Whenever the x value exceeds 80 it is generated as two bytes. If x is less than 81: a (x+bias) (y+bias) If x is greater than 80: a b (x+bias-80) (y+bias)

NOTE

For more information about the values of a, b, and c see the OUTPUT subsection for the CURSOR_POS_BEGIN, CURSOR_POS_SECOND, and CURSOR_POS_THIRD statements.

Statement	Parameter	Description
	BIAS=number	Specifies an integer to be added to the x and y values. The usual number is 32, which is the value of the ASCII space character. The purpose of a bias is to prevent the x and y values from falling in the range of 0 through 31, which have special meanings in communications. This parameter must be used, though it may be zero.
CURSOR_POS_COLUMN_FIRST		VALUE is TRUE if your terminal has a cursor positioning sequence that outputs the column sequence before the row sequence when positioning the cursor. FALSE if your terminal outputs the row before the column (this applies to the binary and ANSI type only).
CURSOR_POS_COLUMN_LENGTH		This is set for ANSI type terminals and only if the terminal sends a set number of bytes to the terminal for column values. If your terminal is not an ANSI type or if it outputs a variable number of decimal bytes, then set VALUE to zero.
CURSOR_POS_ROW_LENGTH		This is set for ANSI type terminals and only if the terminal sends a set number of bytes to the terminal for row values. If your terminal is not an ANSI type or if it outputs a variable number of decimal bytes, then set VALUE to zero.

The following ten statements have either VALUE=TRUE or VALUE=FALSE parameters. These are required parameters.

Statement	Description
HOME_AT_TOP	The CURSOR_HOME sequence sends the cursor to the top left of the screen rather than to the bottom.
HAS_PROTECT	The PROTECT_BEGIN and PROTECT_END sequences can be used to define protected areas on the screen.
MULTIPLE_SIZES	There is more than one SET_SIZE statement.
AUTOMATIC_TABBING	The terminal supports tabbing from one completed, filled, unprotected input field to the next without requiring that a tab key be pressed. FALSE if your terminal does not support protected areas.
TYPE_AHEAD	Allows the Full Screen Editor to run in type ahead mode. This allows you to enter additional input without waiting for the system response to the previous one. Care should be exercised in that type ahead could allow you to make changes that you cannot see on the screen unless you clear the page.
HAS_HIDDEN	The HIDDEN_BEGIN and HIDDEN_END sequences can be used to define areas on the screen in which nothing will be displayed, even if something is typed there.

Statement	Description
TABS_TO_HOME	When the TAB key is pressed and the cursor is on the last unprotected field, the cursor goes to the CURSOR_HOME position rather than wrapping around to the first unprotected field.(The same happens if tabbing backward.) FALSE otherwise or if the terminal does not have protected areas.
TABS_TO_UNPROTECTED	The terminal supports tabbing forward and backward to the start of each unprotected field. False if the terminal does not have protected areas.
TABS_TO_TAB_STOPS	The terminal supports tabbing to settable or predefined tab stops (like typewriter tabs).
CLEARS_WHEN_CHANGE_SIZE	The SET_SIZE sequence causes the screen to be cleared. FALSE if your terminal supports only one screen size.
FUNCTION_KEY_LEAVE_MARK	This is needed for full-screen products to repaint the valid character over the marked area. When a function key is pressed, it causes a character (or characters) to be displayed on the screen, or the use of function keys on the terminal is to be supported by escape or control sequences that require a character to complete the sequence. VALUE is the number of characters that must be erased from the screen after a function key has been pressed. If your terminal leaves no marks when a function key is pressed, VALUE is equal to zero. This statement is required.

CURSOR POSITIONING STATEMENTS

These statements are required. Each has a required TYPE parameter with one of the following values:

Parameter	Description
SCROLL_NEXT	The terminal scrolls all characters on the screen (up, down, or sideways).
STOP_NEXT	The cursor refuses to move beyond the edge.
HOME_NEXT	The cursor moves to the home position.
WRAP_ADJACENT_NEXT	The cursor wraps around to the adjacent line or column at the opposite edge of the screen. For example, if the cursor moves beyond the right edge of the screen, it reappears at the left side on the next line down.
WRAP_SAME_NEXT	The cursor wraps around to the opposite edge of the screen, but in the same line or column. This commonly occurs when the cursor moves beyond the top or bottom. It stays in the same column but at the opposite edge of the screen.

The following statements specify how the terminal behaves when the cursor is urged to go beyond the edge of the screen. Each statement must be included with one of the TYPE parameters listed above.

Statement — Description

MOVE_PAST_LEFT
MOVE_PAST_RIGHT
Describes what happens when the cursor is moved past the left or right edge of the screen by use of the cursor movement keys.

MOVE_PAST_TOP
Describes what happens when the cursor is moved past the top of the screen using the cursor movement keys.

MOVE_PAST_BOTTOM
Describes what happens when the cursor is moved past the bottom of the screen using the cursor movement keys.

CHAR_PAST_LEFT
CHAR_PAST_RIGHT
Describes the action when the cursor moves past the left or right side of the screen because you have typed characters other than the cursor movement keys.

CHAR_PAST_LAST_POSITION
Describes the action when the cursor is moved past the last position on the screen because you typed characters other than the cursor movement keys.

SET SIZE STATEMENT

This statement describes the size or sizes of the terminal screen. It is required for at least one size. If more than one size is specified, you may use the statement up to four times, specifying them in increasing order, giving columns preference over lines.

Statement — Description

SET_SIZE
The sequence specified causes the number of rows and columns to be changed to the values indicated.

Parameters — Description

ROWS=number
The number (an integer) of rows (lines) to which the terminal will be set.

COLUMNS=number
The number (an integer) of columns (characters) to which the terminal will be set.

OUT=sequence
The sequence to be sent to the terminal. This sequence must be obtained from the terminal hardware reference manual.

INITIALIZATION OUTPUT STATEMENTS

Statement	Description
SCREEN_INIT	This sequence is sent whenever the SCREEN command is executed.
LINE_INIT	This sequence is sent whenever the LINE command is executed.
SET_SCREEN_MODE	This sequence is sent whenever the terminal switches from line mode to screen mode.
SET_LINE_MODE	This sequence is sent whenever the terminal switches from screen mode to line mode.

INPUT/OUTPUT STATEMENTS

The following statements define sequences which may be either sent or received by the terminal. All of these statements have a LABEL and an INOUT parameter. Only the INOUT parameter is required.

Statement	Description
INSERT_CHAR	Inserts a single blank character at the current position, shifting present text to the right.
DELETE_CHAR	Deletes a single character at the current position, shifting the present text to the left.
INSERT_LINE_STAY	Inserts a blank line at the current position, the current line shifting down. Leaves the cursor where it is. Only one of the INSERT_LINE_STAY and INSERT_LINE_BOL statements may be used.
INSERT_LINE_BOL	Inserts a blank line at the current position, shifting the current line down. Moves the cursor to the start of the line. Only one of the INSERT_LINE_STAY and INSERT_LINE_BOL statements may be used.
DELETE_LINE_STAY	Deletes the line at the current position, shifting the remaining text up. Leaves the cursor where it is. Only one of the DELETE_LINE_STAY and DELETE_LINE_BOL statements may be used.
DELETE_LINE_BOL	Deletes the line at the current position, shifting the remaining text up. Moves the cursor to the start of the line. Only one of the DELETE_LINE_STAY and DELETE_LINE_BOL statements may be used.
ERASE_PAGE_STAY	Clears the screen, leaving the cursor where it is. One of the ERASE_PAGE_STAY or ERASE_PAGE_HOME is required and only one may be used.

Statement	Description
ERASE_PAGE_HOME	Clears the screen, moving the cursor to the home position. One of the ERASE_PAGE_STAY and ERASE_PAGE_HOME statements is required and only one may be used.
ERASE_UNPROTECTED	Erases all the unprotected character positions on the screen.
ERASE_END_OF_PAGE	Erases the screen from the current cursor position to the bottom of the screen.
ERASE_LINE_STAY	Erases the current line. Leaves the cursor where it is. Only one of the ERASE_LINE_STAY and ERASE_LINE_BOL statements may be used.
ERASE_LINE_BOL	Erases the current line. Moves the cursor to the start of the line. Only one of the ERASE_LINE_STAY and ERASE LINE_BOL statements may be used.
ERASE_END_OF_LINE	Erases from the current position to the end of the line. Leaves the cursor where it is. No full-screen product will function acceptably without this capability.
ERASE_FIELD_STAY	Erases the current unprotected field. Leaves the cursor where it is.
ERASE_FIELD_BOF	Erases the current unprotected field. Moves the cursor to the start of that unprotected field.
ERASE_END_OF_FIELD	Erases from the current position to the end of the unprotected field. Leaves the cursor where it is.
ERASE_CHAR	Erases the character at the current position, moving the cursor left one position.
CURSOR_HOME	Moves the cursor to the home position. No full-screen application will function acceptably without this. This is a required statement.
CURSOR_UP	Moves the cursor up one line. Required statement.
CURSOR_DOWN	Moves the cursor down one line. Required statement.
CURSOR_LEFT	Moves the cursor left one position. Required statement.
CURSOR_RIGHT	Moves the cursor right one position. Required statement.
INSERT_MODE_BEGIN	Enters insert mode. Any graphic characters are inserted, shifting other characters right, rather than overstriking.
INSERT_MODE_END	Exits insert mode. Any graphic characters overstrike rather than insert.

Statement	Description
INSERT_MODE_TOGGLE	Switches between insert and overstrike mode.
BACK_SPACE	Moves the cursor left one position. (This is provided for terminals with a back space key that is unique from the CURSOR_LEFT key.)
TAB_FORWARD	Tabs to the next tab stop or unprotected field.
TAB_BACKWARD	Tabs to the previous tab stop or unprotected field.
TAB_CLEAR	Clears the tab stop at the current position.
TAB_CLEAR_ALL	Clears all tab stops.
TAB_SET	Sets a tab stop at the current position.
RESET	Resets the terminal hardware. The terminal must be reinitialized.

INPUT STATEMENTS

The following statements define character sequences sent by the terminal. They all have an INPUT parameter with values obtained from the terminal hardware reference manual. The first two statements are used to allow direct cursor positioning by the touch panel with the Viking 721 only.

Statement	Description
CURSOR_POS_BEGIN	The first character string of the cursor position sequence. This is a required statement. The value is a in the format.
END_OF_INFORMATION	Signifies end of input. This is a system-dependent, not terminal-dependent statement and the value is normally zero.

CDC Standard Function Keys

All full-screen products use CDC standard function keys. These keys have the same meaning to a particular full-screen product regardless of the terminal in use. The Viking 721 terminal has these CDC standard function keys as actual key caps.

You define what input sequences the terminal you use will send upline to be recognized as a CDC standard function key. This capability will make all full-screen products more usable to the end user but is not required when using the NOS procedures in screen mode.

If local screen formatting applications have been written that use CDC standard function keys (rather than programmable function keys described in the next subsection) to drive menus or to terminate input, then these function keys must be defined in the terminal definition file.

Escape or control sequences such as ESC-H for HELP can be a good way to define CDC standard functions but take care not to use sequences that conflict with terminal hardware sequences.

Unshifted CDC standard function keys:

DOWN
UP
FWD
BKW
NEXT
BACK
STOP
HELP
EDIT
DATA

Shifted CDC standard function keys:

DOWN_S
UP_S
FWD_S
BKW_S
NEXT_S
BACK_S
STOP_S
HELP_S
EDIT_S
DATA_S

Programmable Function Keys

All system-defined full-screen products use programmable function keys to tell the full-screen product what you want to do next. Programmable function keys in the Full Screen Editor allow a frequently used command to execute by pressing one function key or the required sequence of keys for the terminal in use.

You define what input sequences the terminal you use will send upline to be recognized as programmable function keys. These are required parameters for at least the first six keys (F1 through F6) and should be defined for all of the keys if possible for your terminal.

If local screen formatting applications have been written that use programmable function keys to drive menus or to terminate input, then programmable function keys must be defined in the terminal definition file for your terminal.

Escape or control sequences such as ESC-1 for F1 can be a good way to define programmable functions but take care not to use any sequences that conflict with terminal hardware sequences.

Unshifted programmable function keys are:

```
    F1
    f2
    f3
    f4
    f5
    f6
    f7
    f8
    f9
    f10
    f11
    f12
    f13
    f14
    f15
    F16
```

Shifted programmable function keys are:

```
    F1_S
    f2_s
    f3_s
    f4_s
    f5_s
    f6_s
    f7_s
    f8_s
    f9_s
    f10_s
    f11_s
    f12_s
    f13_s
    f14_s
    f15_s
    F16_S
```

OUTPUT STATEMENTS

The following statements define sequences sent to the terminal. Each directive has an OUT parameter that specifies a character string obtained from the terminal hardware reference manual.

Statement	Description
OUTPUT_BEGIN	Send this sequence before starting output (after receiving input). This sequence should include the sequence to disable protected areas if the terminal supports it and also the sequence to exit insert mode if the terminal supports an insert mode.
OUTPUT_END	Send this sequence after ending output (before receiving input). This sequence should include the sequence to enable protected areas if the terminal supports protected areas.
PROTECT_ALL	Every character position on the screen is protected.
RETURN	Move the cursor to the beginning of the current line.
BELL_NAK	Ring the bell on an error. Default is ASCII BEL (7).
BELL_ACK	Ring the alternate bell.
DISPLAY_BEGIN	Enable the display so characters received show on the screen.
DISPLAY_END	Disable the display.
PRINT_BEGIN	Enable the printer so characters received print.
PRINT_END	Disable the printer.

The following statements define character sequences sent by the terminal. They all have an OUTPUT parameter with values obtained from the terminal hardware reference manual. The first three statements are used in conjunction with a CURSOR_POS_ENCODING statement having the axbyc format.

Statement	Description
CURSOR_POS_BEGIN	The first character string of the cursor position sequence. This is a required statement. The value is a in the format.
CURSOR_POS_SECOND	The second character string of the cursor position sequence. This is a required statement if present. The value is b in the format.
CURSOR_POS_THIRD	The third character string of the cursor position sequence. This is a required statement if present. The value is c in the format.

Some terminals actually use a character position on the screen to enable/disable the following attributes. If this is the case with your terminal, do not use the following attributes.

Statement	Description
BLINK_BEGIN	Blinks characters received after this statement.
BLINK_END	Does not blink characters received after this statement.
ALT_BEGIN	Displays characters received after this in alternate intensity (may be bright or dim).
ALT_END	Does not display characters received after this in alternate intensity.
HIDDEN_BEGIN	Does not display characters received after this (sets up "hidden fields", as for passwords).
HIDDEN_END	Displays characters received after this statement.
INVERSE_BEGIN	Displays characters received after this in inverse video.
INVERSE_END	Does not display characters received after this in inverse video.
PROTECT_BEGIN	Makes character positions written to after this protected.
PROTECT_END	Makes character positions written to after this unprotected.
UNDERLINE_BEGIN	Underlines characters received after this statement.
UNDERLINE_END	Does not underline characters received after this statement.

Logical Attribute Statements

Logical attributes are used mainly for procedures executed in screen mode and screen formatting applications to define various types of fields on the screen. Procedures used in screen mode, for example, define all input parameters for a procedure as logical type INPUT TEXT. This assures that they are underlined for those terminals that have that capability or that any blanks in the variables are replaced with hyphen characters on the screen to make them easily recognizable.

You may define the logical attributes below as any combination of physical attributes by using the sequences (obtained from the terminal hardware reference manual) to turn them on and off, or as any other displayable type function that your terminal can support, such as RED_ON for ERROR_BEGIN and RED_END for ERROR_END.

```
INPUT_TEXT_BEGIN
INPUT_TEXT_END
OUTPUT_TEXT_BEGIN
OUTPUT_TEXT_END
ITALIC_BEGIN
ITALIC_END
TITLE_BEGIN
TITLE_END
MESSAGE_BEGIN
MESSAGE_END
ERROR_BEGIN
ERROR_END
```

Line Drawing Statements

Screen formatting applications allow specification of three weights of line drawing (fine, medium, and bold), along with the output sequences for each weight (on and off) and the characters for horizontal lines, vertical lines, box corners, and box intersections.

If your terminal has the capability of actual line drawing, then place the sequences to turn the line drawing on and off in the LD_FINE_BEGIN and LD_FINE_END and so on for up to three types of line drawing sets (you may specify the same sequences for all three or for any two if your terminal has only one or two line drawing sets). If your terminal does not have line drawing then the use of a hyphen character for a horizontal character, a colon or like character for a vertical line, and asterisks for all corners and intersections is recommended. In this case the LD_FINE_BEGIN and LD_FINE_END sequences would be blank though you could use a terminal attribute such as BLINK_ON and BLINK_OFF respectively.

Also for a bold line drawing character set you can define all characters as blanks (' ') and use INVERSE_ON and INVERSE_OFF as the LD_BOLD_BEGIN and LD_BOLD_END sequences.

The following statements can be used to specify line drawings for the three line weights. Different statements specify begin and end, horizontal and vertical lines, the four box corners, and intersection characters. All directives have a required OUT parameter.

```
LS_FINE_BEGIN
LD_FINE_END
LD_FINE_HORIZONTAL
LD_FINE_VERTICAL
LD_FINE_UPPER_LEFT
LD_FINE_UPPER_RIGHT
LD_FINE_LOWER_LEFT
LD_FINE_LOWER_RIGHT
LD_FINE_UP_T
LD_FINE_DOWN_T
LD_FINE_LEFT_T
LD_FINE_RIGHT_T
LD_FINE_CROSS
LD_MEDIUM_BEGIN
LD_MEDIUM_END
LD_MEDIUM_HORIZONTAL
LD_MEDIUM_VERTICAL
LD_MEDIUM_UPPER_LEFT
LD_MEDIUM_UPPER_RIGHT
LD_MEDIUM_LOWER_LEFT
LD_MEDIUM_LOWER_RIGHT
LD_MEDIUM_UP_T
LD_MEDIUM_DOWN_T
LD_MEDIUM_LEFT_T
LD_MEDIUM_RIGHT_T
LD_MEDIUM_CROSS
LD_BOLD_BEGIN
LD_BOLD_END
LD_BOLD_HORIZONTAL
LD_BOLD_VERTICAL
LD_BOLD_UPPER_LEFT
LD_BOLD_UPPER_RIGHT
LD_BOLD_LOWER_LEFT
LD_BOLD_LOWER_RIGHT
LD_BOLD_UP_T
LD_BOLD_DOWN_T
LD_BOLD_LEFT_T
LD_BOLD_RIGHT_T
LD_BOLD_CROSS
```

TDU COMMAND

The TDU command calls an interactive procedure to compile a terminal definition and store the compiled definition in a user library. The compiled output is a load capsule which the procedure stores in a user library.

The user library to receive the load capsule must be a local file. If the library file you specify does not exist as a local file, TDU creates it. If you do not specify a library file, TDU uses a local file with the default name TERMLIB, if one exists. If it does not exist, TDU creates a local file with the name TERMLIB.

In the TDU command format, the parameter keywords and equal signs can be omitted if the parameters are specified in the order listed. The format of the TDU command is:

```
TDU,I=definition,L=listing,LIB=library
```

Parameter	Description
I=definition	Name of the terminal definition file. The file must be in 6/12-bit display code. The I parameter must be specified.
L=listing	Name of the listing file. The listing file is a copy of the input file with error messages (if any) interspersed. The default listing file name is OUTPUT.
LIB=library	Name of the library file to receive the load capsule; must be a local file. The default library name is TERMLIB. To be used by the SCREEN and LINE commands, the library name must be TERMLIB.

Since the TDU command is an interactive procedure, you can receive help information for the procedure and be prompted for parameter entries by entering:

```
TDU?
```

When the SCREEN or LINE command is entered specifying a terminal model name, the command will attempt to locate in file TERMLIB a terminal definition for that model.

Certain terminal definitions have been preloaded into the full-screen products by your installation. If the model you specify is one of these, then SCREEN and LINE look no further.

If the terminal definition is not preloaded by your installation then SCREEN and LINE first look for a local file named TERMLIB, then an indirect access permanent file named TERMLIB under your user name. If such a file exists and contains a definition for the terminal model requested, that definition is used.

If not, SCREEN and LINE look for an indirect file named TERMLIB under user name LIBRARY. Your installation may provide such a file with common terminal definitions in it. If such a file exists and contains a definition for the model requested, that definition is used.

In either of these two cases (a definition is either in your TERMLIB or under user name LIBRARY) SCREEN and LINE copy the definition into a local file named ZZZTERM for later use by the NOS full-screen products. If you see the file, that is what it is for; do not delete it, or you will not be able to run in screen mode until you issue another SCREEN command.

The following example is a terminal definition file for a Viking 721 terminal.

```
"                                                                             "
"   TERMINAL DEFINITION FILE FOR 721 TERMINAL                                 "
"                                                                             "
"        The terminal definition utility (TDU) allows user definition of      "
"   most character mode asynchronous type terminals for use with all NOS      "
"   full screen products.  A detailed description of TDU can be found in      "
"   the NOS Screen Formatting Reference Manual.                               "
"        There should be a collection of system defined terminal defini-      "
"   tions on file TDUFILE under UN=LIBRARY that may assist you (and perhaps   "
"   already define your terminal or one very much like it).  These defini-    "
"   tions and the terminals that they define are:                             "
"                                                                             "
"        TDU721         CDC 721 (Viking)                                      "
"        TDU722         CDC 722                                               "
"        TDUVT10        DEC VT100                                             "
"        TDUT415        TEKTRONIX T4115                                       "
"        TDUZ19         ZENITH Z19/Z29                                        "
"        TDUADM3        LEAR SIEGLER ADM3A                                    "
"        TDUADM5        LEAR SIEGLER ADM5                                     "
"                                                                             "
"        A collection of terminal definition files for other terminals        "
"   will also be made available through Central Software Support for a        "
"   variety of popular terminals and micro computers.                         "
"        This file (TDUIN) is the input file that you will fill with the      "
"   specific terminal dependent data that you should find in the hardware     "
"   reference manual for your terminal.  When the sequences, capabilities     "
"   and attributes of your terminal have been filled in you will then com-    "
"   pile your terminal definition by using the system TDU command.  This      "
"   will produce a file named TERMLIB which contains an encapsulated copy     "
"   of the information needed by NOS screen formatting products to utilize    "
"   your terminal.  The command SCREEN,XXXXXX (XXXXXX being the value you     "
"   specified for the model_name statement) will then enable you to inter-    "
"   act with all NOS full screen facilities.                                  "
"        A number of capabilities are required for your terminal to func-     "
"   tion in screen mode.  These are a clear_page_ stay or a clear_page_       "
"   home, a cursor_home, and the ability to directly position the cursor      "
"   on the screen.  At least a subset (F1-F6) of the application keys and     "
"   a CDC standard STOP function key should also be defined.  An erase_       "
"   end_of_line is not required but will provide considerably better per-     "
"   formance for all full screen products.                                    "
"        Any line surrounded by quotation marks (such as this text) is a      "
"   comment line and will be ignored when compiling your terminal capsule.    "
"   This is a way in which you can add your owm comments to this file as      "
"   you proceed to fill in the requested information.  Those lines that       "
"   are not surrounded by quotation marks in this file are the input dir-     "
"   ectives to TDU for which you will fill in the correct values for your     "
"   terminal.                                                                 "
"        TDU allows you to define variables for commonly used character       "
"   strings and recognizes ASCII mnemonics (such as rs, ack).  Both your      "
"   variables and the mnemonics can be used anywhere in this file.            "
```

Here are some examples to assist you in your definitions:

```
VARIABLES
set_line_mode    = ()          Empty sequence.
set_line_mode    = (rs ack)    ASCII mnemonics.
set_line_mode    = (14(8))     (8) indicates an octal value.
set_line_mode    = (14(16))    (16) indicates a hexadecimal value.
set_line_mode    = (14)        Any nonsubscripted number is decimal.
blank_character  = (' ')       Blank character (see line drawing).
start_underline  = (rs '=')    ASCII mnemonic and character.
stop_underline   = (rs '''')   Use of apostrophe.

clear_all_tabs       = (rs dc2 'Y')
disable_blink        = (eot)
disable_auto_cr      = (rs '''')
disable_protect      = (rs dc2 'L')
enable_auto_cr       = (rs '&')
enable_clear         = (rs '$')
enable_cr_delim      = (rs enq)
enable_blink         = (etx)
enable_protect       = (rs dc2 'K')
enable_typeamatic    = (rs dc2 'i')
end_print            = (rs 7f(16))
large_cyber_mode     = (rs dc2 'B')
page_mode            = (syn)
pop_fn_keys          = (rs dc2 71(16) cr)
push_fn_keys         = (rs dc2 70(16) cr)
scroll_mode          = (dc2)
shift_numeric_pad    = (rs dc2 6B(16))
start_inverse        = (rs 'D')
start_underline      = (ack)
stop_inverse         = (rs 'E')
stop_underline       = (nak)
```

Another use of the TDU capability to define variables can be used to make default function key sequences for FSE (which are also defined in TDUIN) more readable.

```
"         Here is an example for a terminal with a set of six (F1-F6) keys:   "
"                                                                             "
"    VARIABLES FOR FULL SCREEN EDITOR FUNCTION KEY DEFINITIONS                "
"                                                                             "
"    k1      = ('SK1/SM/L/ MARK/;SKS1/SMW/L/MRKCHR/')                         "
"    k2      = ('SK2/MMTP/L/ MOVE/;SKS2/CMTP/L/ COPY/')                       "
"    k3      = ('SK3/IBP/L/ INSB/;SKS3/DB/L/ DELB/')                          "
"    k4      = ('SK4/PF/L/ FIRST/;SKS4/VL/L/ LAST/')                          "
"    k5      = ('SK5/U/L/ UNDO/')                                             "
"    k6      = ('SK6/Q/L/ QUIT/')                                             "
     k1    = ('SK1/SM/L/ MARK/;SKS1/SMW/L/MRKCHR/')
     k2    = ('SK2/MMTP/L/ MOVE/;SKS2/CMTP/L/ COPY/')
     k3    = ('SK3/IBP/L/ INSB/;SKS3/DB/L/ DELB/')
     k4    = ('SK4/PF/L/ FIRST/;SKS4/VL/L/ LAST/')
     k5    = ('SK5/U/L/ UNDO/')
     k6    = ('SK6/Q/L/ QUIT/')
     k7    = ('SK7"L/&?/"L"LOCATE";SK7S/LN/L/LOCNXT/')
     k8    = ('SK8/SVC132/L/132COL/;SK8S/SVC80/L/ 80COL/')
     k9    = ('SK9/V/L/MIDDLE/')
     k10   = ('SK10/.E/L/ENDLIN/')
     k11   = ('SK11/.S/L/ SPLIT/')
     k12   = ('SK12/.J/L/ JOIN/')
     k13   = ('SK13/.F/L/ PARA/')
     k14   = ('SK14/CMTP/L/ COPY/')
     k15   = ('SK15/.C/L/CENTER/')
"                                                                             "
"         There are several basic types of statements that you will en-      "
"    counter in this file:                                                    "
"                                                                             "
"    o    VALUE STATEMENTS                                                    "
"                                                                             "
"         model_name                 value = 'myown'                          "
"         has_protect                value = TRUE                             "
"                                                                             "
"         where VALUE is TRUE, FALSE, an alphabetic string or a number.       "
"                                                                             "
"    o    TYPE STATEMENTS                                                     "
"                                                                             "
"         cursor_pos_encoding        type  = ansi_cursor                      "
"         char_past_last_position    type  = wrap_adjacent_next               "
"                                                                             "
"         where TYPE is one of a predefined list of choices that will         "
"         be listed preceding the statement.                                  "
"                                                                             "
"    o    IN STATEMENTS                                                       "
"                                                                             "
"         f1                         in    = (rs 71(16))                      "
"         help                       in    = (rs 5C(16))                      "
"                                                                             "
"         where IN is the sequence that comes upline from the terminal        "
"         when a specific function is performed or key is pressed.            "
"                                                                             "
"   o    OUT STATEMENTS                                                       "
"                                                                             "
"        cursor_pos_begin         out   = (stx)                               "
"        bell_nak                 out   = (bel)                               "
"                                                                             "
"        where OUT is the sequence sent down line to the terminal to          "
"        perform a certain function.                                          "
"                                                                             "
"   o    INOUT STATEMENTS                                                     "
"                                                                             "
"        erase_page_home          inout = (ff)                                "
"        tab_forward              inout = (ht)                                "
"                                                                             "
"        where INOUT is the identical sequence sent up and down line          "
"        for a certain function.                                              "
"                                                                             "
"        It should be noted that you may break any INOUT statement like       "
"                                                                             "
"        tab_forward              inout = (ht)                                "
"                                                                             "
"        into a matched pair of statements like                               "
"                                                                             "
"        tab_forward              in    = (ht)                                "
"        tab_forward              out   = (ht)                                "
"                                                                             "
"        You will need to do this if your terminal sends a different          "
"        sequence downline to the terminal to perform a certain function      "
"        than is sent upline when that function is performed.                 "
"                                                                             "
"        Any statement that is IN or OUT only should be left as is.           "
"                                                                             "
"        The file from this point on is arranged by functional groups and     "
"   contains comments for each directive that should assist you in fill-      "
"   ing in the correct sequences for your terminal.                           "
"                                                                             "
"   MODEL NAME AND COMMUNICATION TYPE                                         "
"        model_name -  A one to six character alphanumeric name for your      "
"        terminal.  Lower case letters are translated to upper case.  The     "
"        value specified here will be the name used on the SCREEN command.    "
"                                                                             "
    model_name          value = ´721´
"                                                                             "
"        Communication type is asynch as only asynchronous terminals are      "
"        presently supported.                                                 "
"                                                                             "
    communications      type  = asynch
"                                                                             "
"   END OF INFORMATION SPECIFICATION                                          "
"        This defines the end of information sequence which is a zero byte.   "
"                                                                             "
    end_of_information  in    = (0)
"                                                                             "
"   CURSOR POSITIONING INFORMATION                                            "
"                                                                             "
"       The way in which your terminal encodes cursor positioning will      "
"       determine your choice for cursor_pos_encoding and cursor_pos_       "
"       column_first.  The general format for cursor positioning is:        "
"                                                                           "
"       Let X     ---------------> represent the column coordinate.         "
"       Let Y     ---------------> represent the row coordinate.            "
"       Let a     ---------------> represent cursor_pos_begin.              "
"       Let b     ---------------> represent cursor_pos_second.             "
"       Let c     ---------------> represent cursor_pos_third.              "
"       And Bias  ---------------> is the integer value added to the        "
"                                  row or column for cursor positioning.    "
"                                  You should be able to find the value     "
"                                  for bias in the harware reference man-   "
"                                  ual for your terminal (often 20(16)).    "
"                                                                           "
"       Then cursor_pos_encoding will be one of three type:                 "
"                                                                           "
"           ansi_cursor    ----> Those terminals which are ansi standard    "
"                                and use decimalized cursor coordinates.    "
"                                Format is:                                 "
"                                    a (X + bias) b (Y + bias) c            "
"                                    a (Y + bias) b (X + bias) c            "
"                                the order of X and Y for your terminal     "
"                                determines the value for cursor_pos_       "
"                                column_first.                              "
"                                                                           "
"           cdc721_cursor ----> The Control Data 721 (Viking X) terminal.   "
"                                Format is:                                 "
"                                    a   (X + bias)      (Y + bias)         "
"                                        (if X is less than 81)             "
"                                    a b (X + bias -80) (Y + bias)          "
"                                        (if X greater than 80)             "
"                                                                           "
"           binary_cursor ----> Those terminals which use direct co-        "
"                                ordinate positioning.                      "
"                                Format is:                                 "
"                                    a (X + bias) b (Y + bias) c            "
"                                    a (Y + bias) b (X + bias) c            "
"                                the order of X and Y for your terminal     "
"                                determines the value for cursor_pos_       "
"                                column_first.                              "
"                                                                           "
    cursor_pos_encoding      bias  = (32)   type = cdc721_cursor
"                                                                           "
"       Cursor_pos_column_first has a value of TRUE if your terminal        "
"       sends the X (or column) coordinate followed by the Y (or row)       "
"       coordinate and has a value of FALSE if the reverse is true.         "
"                                                                           "
    cursor_pos_column_first  value = TRUE
```

```
"                                                                       "
"       Cursor_pos_column_length and row_length apply only to ANSI type  "
"       cursor position (there are zero for both other types) and are    "
"       non-zero only if your terminal sends a fixed number of decimal-  "
"       ized bytes for the column and row coordinates (as opposed to a   "
"       variable number which is the usual case).                        "
"                                                                       "
    cursor_pos_column_length value = (0)
    cursor_pos_row_length    value = (0)
"                                                                       "
"       Cursor_pos_begin, second and third are the sequences sent before "
"       the first coordinate, in between coordinates and after the last  "
"       coordinate when positioning the cursor (a b and c in the formats "
"       shown above).  At least a cursor_pos_begin should be supplied    "
"       for your terminal though second and third are often an empty     "
"       sequence and can be left alone.                                  "
"                                                                       "
    cursor_pos_begin         out   = (stx)
    cursor_pos_second        out   = (7E(16) soh)
    cursor_pos_third         out   = ()
"                                                                       "
"   CURSOR MOVEMENT INFORMATION                                          "
"                                                                       "
"       Cursor_home, up, down, left and right are the sequences sent both "
"       downline to the terminal to move the cursor to the home position "
"       or a single column or row up, down, left, or right and upline    "
"       from the terminal when a cursor key is pressed.  Since this is   "
"       both and upline and downline sequence the INOUT keyword is used. "
"                                                                       "
    cursor_home              inout = (em)
    cursor_up                inout = (etb)
    cursor_down              inout = (sub)
    cursor_left              inout = (bs)
    cursor_right             inout = (can)
"                                                                       "
"   CURSOR BEHAVIOR (for cursor movement keys)                           "
"                                                                       "
"       Move_past_right, left, top and bottom describe when happens      "
"       when the cursor on your terminal is urged to move past the       "
"       right, left, top and bottom of the screen by a cursor move-      "
"       ment key (not by cursor movement caused by character input       "
"       or a seperate backspace key your terminal may have in add-       "
"       ition to a cursor left key, these behaviors may be different     "
"       from those for cursor positioning keys and will be defined       "
"       in the next section).  The possible types are:                   "
"                                                                       "
"            wrap_adjacent_next ----> The cursor wraps to the other end  "
"                                     of the screen on the adjacent row  "
"                                     (next row cursor_right or previous "
"                                     row for cursor_left)               "
"            wrap_same_next     ----> The cursor wraps to the other      "
"                                     end of the screen still in the     "
"                                     same row or column.                "
"            scroll_next        ----> The terminal scrolls.              "
"            stop_next          ----> The cursor stops                   "
"            home_next          ----> The cursor homes.                  "
"                                                                            "
    move_past_right         type  = wrap_adjacent_next
    move_past_left          type  = wrap_adjacent_next
    move_past_top           type  = wrap_same_next
    move_past_bottom        type  = wrap_same_next
"                                                                            "
"   CURSOR BEHAVIOR (for character keys)                                     "
"                                                                            "
"        Char_past_right, left and last_postion describe when happens        "
"        when the cursor on your terminal is urged to move past the          "
"        right, left and end of the screen by character input or a           "
"        seperate backspace key your terminal has in addition to (or         "
"        in place of) a cursor left key.  The possible behaviors are         "
"        the same as those for cursor positioning keys.                      "
"                                                                            "
"             wrap_adjacent_next ----> The cursor wraps to the other end     "
"                                      of the screen on the adjacent row     "
"                                      (next row cursor_right or previous    "
"                                      row for cursor_left)                  "
"             wrap_same_next     ----> The cursor wraps to the other         "
"                                      end of the screen still in the        "
"                                      same row or column.                   "
"             scroll_next       ---->  The terminal scrolls.                 "
"             stop_next         ---->  The cursor stops                      "
"             home_next          ----> The cursor homes.                     "
"                                                                            "
    char_past_right         type  = wrap_adjacent_next
    char_past_left          type  = wrap_adjacent_next
    char_past_last_position type  = wrap_adjacent_next
"                                                                            "
"   TERMINAL ATTRIBUTES                                                      "
"                                                                            "
"        These describe various attributes and capabilites of your          "
"        terminal that should be either TRUE or FALSE.                       "
"                                                                            "
"        Automatic_tabbing is TRUE if your terminal supports tabbing         "
"        from one completed filled unprotected input field to the            "
"        next without requiring that a tab key is pressed.                   "
"                                                                            "
    automatic_tabbing       value = FALSE
"                                                                            "
"        Clears_when_change_size is TRUE if your terminal has more than      "
"        one screen size and changing screen sizes causes the screen to      "
"        be cleared.                                                         "
"                                                                            "
    clears_when_change_size value = TRUE
"                                                                            "
"        Function_key_leaves_mark is TRUE if pressing a function key         "
"        on your terminal leaves a visible mark or character on the          "
"        screen or if function keys for your terminal will be support-       "
"        ed by an escape or control sequence that will require a char-       "
"        acter to complete.  The full screen editor will then know to        "
"        rewrite the line on the screen that has been overwritten by         "
"        the mark or character(s).                                           "
"                                                                            "
    function_key_leaves_mark value = FALSE
```

```
"                                                                              "
"        Has_hidden is TRUE if your terminal supports a hidden attribute       "
"        that allows a field to be defined as input only such that typed       "
"        characters are not displayed on the screen.                           "
"                                                                              "
    has_hidden               value = TRUE
"                                                                              "
"        Has_protect is TRUE if the terminal hardware supports a protected     "
"        field attribute so that users can only enter data within specified   "
"        areas on the screen.                                                  "
"                                                                              "
    has_protect              value = TRUE
"                                                                              "
"        Home_at_top is TRUE if the cursor goes to the top of the screen       "
"        when the home key is pressed or FALSE if it goes to the bottom.       "
"                                                                              "
    home_at_top              value = TRUE
"                                                                              "
"        Multiple_sizes is true if your terminal has more than one screen      "
"        size that can be set by a sequence sent downline to the terminal.     "
"                                                                              "
    multiple_sizes           value = TRUE
"                                                                              "
"        Tabs_to_home is TRUE if when tabbing forward from the last un-        "
"        protected field on the screen (or backward from the first) the        "
"        cursor moves to the home position and will move to the field          "
"        when the tab key is pressed again.  Set FALSE if your terminal        "
"        can tab directly from the last unprotected field to the first         "
"        (and vice versa) or if your terminal does not support a pro-          "
"        tect attribute.                                                       "
"                                                                              "
    tabs_to_home             value = FALSE
"                                                                              "
"        Tabs_to_tab_stops is TRUE if your terminal supports hardware          "
"        tabbing to tab stops, FALSE otherwise.                                "
"                                                                              "
    tabs_to_tab_stops        value = TRUE
"                                                                              "
"        Tabs_to_unprotected is TRUE if your terminal supports tabbing         "
"        from one unprotected field to the next (or previous).  Set to         "
"        FALSE if the terminal does not support protect or protected           "
"        tabbing.                                                              "
"                                                                              "
    tabs_to_unprotected      value = FALSE
"                                                                              "
"        Type_ahead is TRUE if you wish to run the full screen editor          "
"        in type ahead mode, FALSE if you do not.  This has no effect          "
"        on screen formatting applications.  Type ahead means that you         "
"        do not have to wait for the system response to each carriage          "
"        return (next key) but may continue to type.  Care should be           "
"        exercised not to abuse this capability since it is possible           "
"        to produce a screen that does not reflect the actual file             "
"        contents.  If you fear this is the case do a clear page or            "
"        a SET SCREEN (SS) command to tell FSE to repaint the screen.          "
"        In addition typed ahead control t-s (STOP keys) can not pre-          "
"        sently be handled by FSE so you should avoid using procedures         "
"        unless you are sure they will end and not loop continuously.          "
"                                                                              "
    type_ahead               value = FALSE
```

```
"                                                                  "
"  SCREEN SIZES                                                    "
"                                                                  "
"       These sequences are those necessary to set the terminal to a    "
"       specific number of lines and columns if the terminal has more   "
"       than one screen size that can be downline configured.  If the   "
"       terminal does have more than one size specify them in ascend-   "
"       ing order (giving columns preference over lines) by duplicat-   "
"       ing the entire set_size rows = yy columns = xx out = (sequence) "
"       statement.  A maximum of four sizes and a minimum of one are    "
"       to be specified.                                                "
"                                                                  "
"       Rows is the integer number of rows (lines) on the screen for    "
"       a specific screen size.                                         "
"                                                                  "
"       Columns is the integer number of columns (characters per line)  "
"       for a specified screen size.                                    "
"                                                                  "
"       Out is the sequence to be sent to the terminal to set a the     "
"       screen size (it may be an empty sequence for a terminal with    "
"       only one size but the rows and columns should still be entered)."
"                                                                  "
   set_size       rows = 30 columns = 80   out = (rs dc2 'H' rs dc2 '^')
   set_size       rows = 30 columns = 132  out = (rs dc2 'G' rs dc2 '^')
"                                                                  "
"  SCREEN AND LINE MODE TRANSITION                                 "
"                                                                  "
"       Screen_init is the sequence that will be sent to the terminal   "
"       when a SCREEN,TERMINAL_NAME command (or a SCREEN. command when  "
"       a SCREEN,TERMINAL_NAME or LINE,TERMINAL_NAME identifying the    "
"       terminal has previously been executed) is executed.  This is    "
"       useful for a terminal that requires a large amount of recon-    "
"       figuration, some of which does not affect line mode dialogs     "
"       and thus does not have to be done at each entrance to a full    "
"       screen application (see set_screen_mode).                       "
"                                                                  "
   screen_init         out = ()
"                                                                  "
"       Line_init is the sequence that will be sent to the terminal     "
"       when a LINE,TERMINAL_NAME command (or a LINE. command when      "
"       a SCREEN,TERMINAL_NAME or LINE,TERMINAL_NAME identifying the    "
"       terminal has previously been executed) is executed.             "
"                                                                  "
   line_init           out = ()
"                                                                  "
"       Set_screen_mode is the sequence that will be sent when the      "
"       terminal enters the full screen editor or a screen formatting   "
"       application.  This is where page mode should be set, tabs per-  "
"       haps cleared and so on to configure for running is screen mode. "
"                                                                  "
   set_screen_mode     out = (push_fn_keys  shift_numeric_pad  enable_clear...
       large_cyber_mode  disable_auto_cr  enable_cr_delim  clear_all_tabs ...
       enable_blink  end_print  page_mode)
```

```
"                                                                     "
"        Set_line_mode is the sequence that will be sent when the     "
"        terminal exits the full screen editor or a screen formatting "
"        application.  This is where roll (or line) should be set and "
"        what was done by the set_screen_mode sequence reversed.      "
"                                                                     "
    set_line_mode       out = (scroll_mode  enable_auto_cr  clear_all_tabs ...
        pop_fn_keys)
"                                                                     "
"   TERMINAL CAPABILITIES                                             "
"                                                                     "
"        These define what capabilities such as local insert and      "
"        delete line or character your terminal provides.             "
"                                                                     "
"        Backspace allows you to define a key that sends a different  "
"        (from the cursor left key) sequence upline from the terminal "
"        to move the cursor one character position to the left.  This"
"        is of particular use if the behavior for the backspace key   "
"        (which will be treated as a character movement key, not a    "
"        cursor movement key and hence is bound by the CHARACTER MOVE-"
"        MENT BEHAVIOR descriptions) differs from the CURSOR MOVEMENT "
"        BEHAVIOR for the cursor_left key (as described in the CURSOR "
"        MOVEMENT BEHAVIOR section of this file).  This is an input   "
"        only sequence so the IN keyword is used here.                "
"                                                                     "
    backspace           in    = ()
"                                                                     "
"        Delete_char is the sequence for local delete character for   "
"        your terminal.  In order for this to function correctly the "
"        key that does the local (that is on the screen) delete char- "
"        acter must send a sequence upline to make the full screen    "
"        product aware that the screen has changed.  This is true     "
"        for all terminal capabilities.                               "
"                                                                     "
    delete_char         inout = (rs 4e(16))
"                                                                     "
"        Delete_line_bol and delete_line_stay are provided so that    "
"        full screen applications are aware of the cursor position    "
"        after a delete line function has been performed.  If your    "
"        terminal has a local delete line function then one (and      "
"        only one) of delete_line_bol or delete_line_stay should      "
"        be filled with the correct terminal sequence.  Delete_line_  "
"        bol if the cursor moves to the leftmost position when a      "
"        line is deleted, delete_line_stay if the cursor stays in     "
"        the column it was in when the delete line function was per-  "
"        formed.                                                      "
"                                                                     "
    delete_line_bol     inout = ()
    delete_line_stay    inout = (rs 51(16))
"                                                                     "
"        Erase_char is the sequence for an erase character function.  "
"                                                                     "
    erase_char          inout = (1f(16))
```

```
"                                                                    "
"       Erase_end_of_line is the sequence for an erase from the      "
"       current cursor position to the end of that line.  This is    "
"       not a required terminal capability but will provide much     "
"       better performance for all full screen products.             "
"                                                                    "
    erase_end_of_line   inout = (vt)
"                                                                    "
"       Erase_field_bof is reserved for future use.                  "
"                                                                    "
    erase_field_bof     inout = ()
"                                                                    "
"       Erase_field_stay is reserved for future use.                 "
"                                                                    "
    erase_field_stay    inout = ()
"                                                                    "
"       Erase_line_bol and erase_line_stay are provided so that      "
"       full screen applications are aware of the cursor position    "
"       after a erase line function has been performed.  If your     "
"       terminal has a local erase line function then one (and       "
"       only one) of erase_line_bol or erase_line_stay should        "
"       be filled with the correct terminal sequence.  Erase_line_   "
"       bol if the cursor moves to the leftmost position when a      "
"       line is erased, erase_line_stay if the cursor stays in       "
"       the column it was in when the erase line function was per-   "
"       formed.                                                      "
"                                                                    "
    erase_line_bol      inout = (rs 5D(16))
    erase_line_stay     inout = ()
"                                                                    "
"       Erase_page_home and erase_page_stay are provided so that     "
"       full screen applications are aware of the cursor position    "
"       after an erase page function has been performed.  If your    "
"       terminal has a local erase page function (that sends a       "
"       a sequence upline) then one (and only one) of erase_page_    "
"       home or erase_page_stay should be filled with the correct    "
"       terminal sequence.  Erase_page_home if the cursor moves to   "
"       the home position when the screen is cleared, erase_page_    "
"       stay if the cursor stays where it was when the erase page    "
"       function was performed.                                      "
"                                                                    "
    erase_page_home     inout = (ff)
    erase_page_stay     inout = ()
"                                                                    "
"       Insert_char is the sequence for local insert character for   "
"       your terminal.  In order for this to function correctly the  "
"       key that does the local (that is on the screen) insert char- "
"       acter must send a sequence upline to make the full screen    "
"       product aware that the screen has changed.  This is true     "
"       for all terminal capabilities.                               "
"                                                                    "
    insert_char         inout = (rs 4f(16))
```

```
"                                                                   "
"        Insert_line_bol and insert_line_stay are provided so that  "
"        full screen applications are aware of the cursor position  "
"        after a insert line function has been performed.  If your  "
"        terminal has a local insert line function (that sends a    "
"        a sequence upline) then one (and only one) of insert_line_ "
"        bol or insert_line_stay should be filled with the correct  "
"        terminal sequence.  Insert_line_bol if the cursor moves to "
"        the leftmost position when a line is inserted, insert_line_"
"        stay if the cursor stays in the column it was in when the  "
"        insert line function was performed.                        "
"                                                                   "
    insert_line_bol     inout = ()
    insert_line_stay    inout = (rs 52(16))
"                                                                   "
"        Erase_unprotected is reserved for future use.              "
"                                                                   "
    erase_unprotected   inout = ()
"                                                                   "
"        Erase_end_of_page is reserved for future use.              "
"                                                                   "
    erase_end_of_page   inout = ()
"                                                                   "
"        Erase_end_of_field is reserved for future use.             "
"                                                                   "
    erase_end_of_field  inout = ()
"                                                                   "
"        Insert_mode_begin is the sequence to enter insert mode.  Char-  "
"        acters are inserted, shifting other characters right rather     "
"        than overstriking them.                                    "
"                                                                   "
    insert_mode_begin   inout = ()
"                                                                   "
"        Insert_mode_end is the sequence to exit insert mode.  Characters "
"        will now overstrike rather than insert.                    "
"                                                                   "
    insert_mode_end     inout = ()
"                                                                   "
"        Insert_mode_toggle will switch between insert and overstike "
"        mode.                                                      "
"                                                                   "
    insert_mode_toggle  inout = ()
"                                                                   "
"        Tab_backward is the sequence sent (and received) when tabbing    "
"        from a tab stop or unprotected field to the previous tab stop    "
"        or unprotected field.                                      "
"                                                                   "
    tab_backward        inout = (rs 0b(16))
"                                                                   "
"        Tab_clear is the sequence to clear the tab stop at the current   "
"        cursor position.                                           "
"                                                                   "
    tab_clear           inout = (rs dc2 ´X´)
"                                                                   "
"        Tab_clear_all is the sequence to clear all tab stops.      "
"                                                                   "
    tab_clear_all       inout = (clear_all_tabs)
```

```
"                                                                             "
"        Tab_forward is the sequence sent (and received) when tabbing         "
"        from a tab stop or unprotected field to the next tab stop or         "
"        unprotected field.                                                   "
"                                                                             "
    tab_forward         inout = (ht)
"                                                                             "
"        Tab_set is the sequence to set a tab stop at the current cursor      "
"        position.                                                            "
"                                                                             "
    tab_set             inout = (rs dc2 'W')
"                                                                             "
"   MISCELLANEOUS TERMINAL SEQUENCES                                          "
"                                                                             "
"        Bell_nak is the sequence to ring the bell on your terminal.          "
"                                                                             "
    bell_nak            out = (bel)
"                                                                             "
"        Bell_ack is reserved for future use.                                 "
"                                                                             "
    bell_ack            out = ()
"                                                                             "
"        Display_begin is reserved for future use.                            "
"                                                                             "
    display_begin       out = ()
"                                                                             "
"        Display_end is reserved for future use.                              "
"                                                                             "
    display_end         out = ()
"                                                                             "
"        Field_scroll_down is reserved for future use.                        "
"                                                                             "
    field_scroll_down   out = ()
"                                                                             "
"        Field_scroll_set is reserved for future use.                         "
"                                                                             "
    field_scroll_set    out = ()
"                                                                             "
"        Field_scroll_up is reserved for future use.                          "
"                                                                             "
    field_scroll_up     out = ()
"                                                                             "
"        Output_begin is the sequence that will be sent before each           "
"        stream of output is sent downline to the terminal.  This             "
"        should include the sequence to disable protect if the term-          "
"        inal supports it as well as the sequence to exit insert mode         "
"        if the terminal has an insert mode.                                  "
"                                                                             "
    output_begin        out = (disable_protect)
"                                                                             "
"        Output_end is the sequence that will be sent after each stream       "
"        of output (and therefore before the next request for input) is       "
"        sent downline to the terminal.  This should include the seq-         "
"        uence to enable protect if the terminal supports protect.            "
"                                                                             "
    output_end          out = (enable_protect)
```

```
"                                                                             "
"        Print_begin is reserved for future use.                              "
"                                                                             "
    print_begin         out = ()
"                                                                             "
"        Print_end is reserved for future use.                                "
"                                                                             "
    print_end           out = ()
"                                                                             "
"        Print_page is reserved for future use.                               "
"                                                                             "
    print_page          out = ()
"                                                                             "
"        Protect_all is the sequence that will set the protect bit for        "
"        all characters positions on the screen.  For some terminals          "
"        that have protect this will be an empty string (an example is        "
"        is a terminal that uses a clear screen to protected character        "
"        positions sequence to accomplish this function).                     "
"                                                                             "
    protect_all         out = (rs 'G')
"                                                                             "
"        Reset is reserved for future use.                                    "
"                                                                             "
    reset               out = ()
"                                                                             "
"        Return is reserved for future use.                                   "
"                                                                             "
    return              out = ()
"                                                                             "
"   PROGRAMMABLE FUNCTION KEY INPUT INFORMATION                               "
"                                                                             "
"   All full screen products use programmable function keys so that a        "
"   user can tell the full screen product what they want to do next.         "
"   Programmable function keys in the full screen editor allow a fre-         "
"   quently used command to be reduced to pressing the correct func-          "
"   tion key (or required sequence of keys) for the terminal in use.          "
"                                                                             "
"   This section allows you to define what input sequences will be sent       "
"   upline by your terminal to be recognized as programmable function         "
"   keys.  These should be entered for at least F1 - F6 and should be         "
"   defined for all of the keys if possible.                                  "
"                                                                             "
"   Procedures run in screen mode will require only F1 - F6 to execute        "
"   correctly but local screen formatting application programs that use       "
"   programmable function keys to drive menus or to terminate form type       "
"   input may require that more programmable functions keys than just        "
"   F1 - F6 be defined in this file.                                          "
"                                                                             "
"   Escape or control sequences such as ESC - 1 for F1 can be a good          "
"   way to define programmable function keys but take care not to use        "
"   sequences that conflict with terminal hardware sequences.  These          "
"   are input only sequences so the IN keyword is used here.                  "
"                                                                      "
    f1        in = (rs 71(16))
    f2        in = (rs 72(16))
    f3        in = (rs 73(16))
    f4        in = (rs 74(16))
    f5        in = (rs 75(16))
    f6        in = (rs 76(16))
    f7        in = (rs 77(16))
    f8        in = (rs 78(16))
    f9        in = (rs 79(16))
    f10       in = (rs 7A(16))
    f11       in = (rs 7B(16))
    f12       in = (rs 7C(16))
    f13       in = (rs 7D(16))
    f14       in = (rs 7E(16))
    f15       in = (rs 70(16))
    f16       in = (rs dc2 31(16))
    f1_s      in = (rs 61(16))
    f2_s      in = (rs 62(16))
    f3_s      in = (rs 63(16))
    f4_s      in = (rs 64(16))
    f5_s      in = (rs 65(16))
    f6_s      in = (rs 66(16))
    f7_s      in = (rs 67(16))
    f8_s      in = (rs 68(16))
    f9_s      in = (rs 69(16))
    f10_s     in = (rs 6A(16))
    f11_s     in = (rs 6B(16))
    f12_s     in = (rs 6C(16))
    f13_s     in = (rs 6D(16))
    f14_s     in = (rs 6E(16))
    f15_s     in = (rs 60(16))
    f16_s     in = (rs dc2 32(16))
"                                                                      "
"   CDC STANDARD FUNCTION KEY INPUT INFORMATION                        "
"                                                                      "
"   All full screen products use what are called CDC standard func-   "
"   tion keys. These are keys that have the same meaning to a par-     "
"   ticular full screen product regardless of the terminal in use.     "
"   Each of these keys also corresponds to a physical key on the       "
"   CDC 721 (Viking) terminal.                                         "
"                                                                      "
"   The next section allows you to define what input sequences the     "
"   terminal you wish to use will send upline to be recognized as      "
"   CDC standard function keys.  This capability will make all full    "
"   screen products more usable to the end user but is not required    "
"   when using the Full Screen Editor or Procedures in screen mode.    "
"                                                                      "
"   Local screen formatting applications that have been written to use "
"   CDC standard function keys (rather than programmable function keys "
"   described in the previous section) to drive menus or to terminate  "
"   form type input may require that at least some CDC standard function"
"   keys be defined in this file.                                      "
"                                                                      "
"   Escape or control sequences such as ESC - F for Forward can be     "
"   a good way to define CDC standard function keys but take care not  "
"   to use sequences that conflict with terminal hardware sequences.   "
"   These are input only sequences so the IN keyword is used here.     "
"                                                                    "
   back       in = (rs 5F(16))
   back_s     in = (rs 5B(16))
   help       in = (rs 5C(16))
   help_s     in = (rs 58(16))
   stop       in = (rs 49(16))
   stop_s     in = (rs 4A(16))
   down       in = (rs dc2 20(16))
   down_s     in = (rs dc2 21(16))
   up         in = (rs dc2 24(16))
   up_s       in = (rs dc2 25(16))
   fwd        in = (rs dc2 28(16))
   fwd_s      in = (rs dc2 29(16))
   bkw        in = (rs dc2 2C(16))
   bkw_s      in = (rs dc2 2d(16))
   edit       in = (rs 5E(16))
   edit_s     in = (rs 5A(16))
   data       in = (rs dc2 35(16))
   data_s     in = (rs dc2 36(16))
"                                                                    "
"  TERMINAL VIDEO ATTRIBUTES                                         "
"                                                                    "
"  These attributes are used mainly by screen formatting applications"
"  to define various types of fields (though protect_begin and end as"
"  well as inverse_begin and end or alternate_begin and end where they"
"  available are used by FSE).                                       "
"                                                                    "
"  Define the attributes sequences below as described in the hardware"
"  reference manual for your terminal.  The only restriction is that "
"  attributes that require an actual character position on the screen"
"  can not be used.  If your terminal has a protect mode that uses a "
"  video attribute such as alternate video (either bright or dim) then"
"  you will want to place these sequences in the protect_begin and pro-"
"  tect_end statements.  These sequences are output only hence the OUT"
"  keyword is used here.                                             "
"                                                                    "
"       Alt_begin is the sequence to cause subsequent characters sent"
"       downline to be displayed in an alternate intensity (which may"
"       be bright or dim on your terminal).                          "
"                                                                    "
   alt_begin            out = (fs)
"                                                                    "
"       Alt_end is the sequence to cause subsequent characters sent  "
"       downline to be in normal intensity.                          "
"                                                                    "
   alt_end              out = (gs)
"                                                                    "
"       Blink_begin is the sequence to cause subsequent characters   "
"       sent downline to be displayed with a blinking attribute.     "
"                                                                    "
   blink_begin          out = (so etx)
"                                                                    "
"       Blink_end is the sequence to cause subsequent characters     "
"       sent downline after this with not be displayed with the      "
"       blinking attribute.                                          "
"                                                                    "
   blink_end            out = (si)
```

```
"                                                                             "
"        Hidden_begin is the sequence to set the hidden attribute for         "
"        subsequent characters so that data typed in this area can not        "
"        be seen on the screen (also called a guarded attribute).             "
"                                                                             "
    hidden_begin        out = (rs dc2 '[')
"                                                                             "
"        Hidden_end is the sequence to return to visible characters.          "
"                                                                             "
    hidden_end          out = (rs dc2 5C(16))
"                                                                             "
"        Inverse_begin is the sequence to cause subsequent characters         "
"        to be displayed in inverse video.                                    "
"                                                                             "
    inverse_begin       out = (start_inverse)
"                                                                             "
"        Inverse_end is the sequence to return to normal video.               "
"                                                                             "
    inverse_end         out = (stop_inverse)
"                                                                             "
"        Protect_begin is the sequence to cause subsequent characters         "
"        sent downline to the terminal to be protected, which means           "
"        data can not be typed in these character positions on the           "
"        screen.                                                              "
"                                                                             "
    protect_begin       out = (rs dc2 'I')
"                                                                             "
"        Protect_end is the sequence to return to unprotected mode.           "
"                                                                             "
    protect_end         out = (rs dc2 'J')
"                                                                             "
"        Underline_begin is the sequence to cause subsequent characters       "
"        sent downline to be displayed with an underline attribute.           "
"                                                                             "
    underline_begin     out = (start_underline)
"                                                                             "
"        Underline_end is the sequence to cause subsequent characters         "
"        sent downline to no longer be underlined.                            "
"                                                                             "
    underline_end       out = (stop_underline)
"                                                                             "
"   LOGICAL ATTRIBUTE SPECIFICATIONS                                          "
"                                                                             "
"   Logical attributes are used mainly by screen formatting applications      "
"   to define various types of fields. Procedures run in screen mode for      "
"   example define all input variables for a procedure as logical type        "
"   INPUT TEXT which assures that they are underlined for those terminals     "
"   that have that capability or that any blanks in the variables are rep-    "
"   laced with hypen characters on the screen to make them easily recogniz-   "
"   able.                                                                     "
"                                                                             "
"   You may define the logical attributes below as any combination of phy-    "
"   sical attributes by using the sequences to turn them on and off or use    "
"   any other displayable type function (except an attribute that will re-    "
"   quire a actual character position on the screen) that your terminal      "
"   supports, such as RED_ON for error_begin and RED_OFF for error_end.       "
"                                                                              "
"   ERROR                                                                      "
"                                                                              "
    error_begin         out = (start_inverse)
    error_end           out = (stop_inverse)
"                                                                              "
"   INPUT TEXT                                                                 "
"        If your terminal supports protect by use of a video attribute         "
"   such as alternate intensity for unprotected areas of the screen you        "
"   should define input_text_begin and end accordingly so that screen          "
"   formatting applications display the input fields correctly as un-          "
"   protected areas.                                                           "
"                                                                              "
    input_text_begin    out = (start_underline)
    input_text_end      out = (stop_underline)
"                                                                              "
"   ITALIC                                                                     "
"        If your terminal supports an alternate character set then here        "
"   is a place that you can make use of it with screen formatting app-         "
"   lications.                                                                 "
"                                                                              "
    italic_begin        out = ()
    italic_end          out = ()
"                                                                              "
"   MESSAGE                                                                    "
"        Attributes display here will be used when printing help and           "
"   error messages on the first line of the screen when a screen for-          "
"   matting application is running.  Use any physical attributes that         "
"   you wish but remember that if your terminal has a video attribute          "
"   based protect capability this area should be protected data.               "
"                                                                              "
    message_begin       out = ()
    message_end         out = ()
"                                                                              "
"   OUTPUT TEXT                                                                "
"        For output only data so if your terminal has a video attribute        "
"   based protect capability this area should be protected data.               "
"                                                                              "
    output_text_begin   out = ()
    output_text_end     out = ()
"                                                                              "
"   TITLE                                                                      "
"                                                                              "
    title_begin         out = ()
    title_end           out = ()
"                                                                              "
"   LINE DRAWING CHARACTER SPECIFICATION                                       "
"                                                                              "
"        Line drawing character sets that your terminal supports should        "
"   be specified here for use with the box drawing capabilty found in          "
"   NOS screen formatting.  There are three line weights, fine, medium,        "
"   and bold, each with a sequence to enable and disable that weight           "
"   and with eleven characters that represent the corners, edges and           "
"   intersections for the corresponding line drawing character set.           "
"                                                                         "
"        If your terminal has the capability of actual line drawing       "
"   then place the sequences to turn the line drawing on and off in       "
"   the ld_fine_begin, ld_fine_end and so on for up to three types of     "
"   line drawing sets (you may specify the same sequences for all three   "
"   or for any two if your terminal does not have three line drawing      "
"   sets).  If your terminal has no line drawing then the use of a        "
"   hypen character for a horizontal character, a colon or like char-     "
"   acter for a vertical line, and asterisks for all corners and in-      "
"   tersections is suggested.  In this case the ld_fine_begin, ld_        "
"   fine_end sequences would be blank though you could use a terminal     "
"   attribute such as alternate intensity.                                "
"                                                                         "
"        Also for a bold line drawing character set you can define        "
"   all characters as blanks (´ ´) and use inverse_on and inverse_off     "
"   as the ld_bold_begin and ld_bold_end sequences.                       "
"                                                                         "
"        Fine Line Drawing.                                               "
"                                                                         "

    ld_fine_begin             out = (rs fs)
    ld_fine_end               out = (rs gs)
    ld_fine_horizontal        out = 20(16)
    ld_fine_vertical          out = 21(16)
    ld_fine_upper_left        out = 22(16)
    ld_fine_upper_right       out = 23(16)
    ld_fine_lower_left        out = 24(16)
    ld_fine_lower_right       out = 25(16)
    ld_fine_up_t              out = 26(16)
    ld_fine_down_t            out = 27(16)
    ld_fine_left_t            out = 28(16)
    ld_fine_right_t           out = 29(16)
    ld_fine_cross             out = 2A(16)
"                                                                         "
"        Medium Line Drawing.                                             "
"                                                                         "

    ld_medium_begin           out = (rs fs)
    ld_medium_end             out = (rs gs)
    ld_medium_horizontal      out = 2B(16)
    ld_medium_vertical        out = 2C(16)
    ld_medium_upper_left      out = 2D(16)
    ld_medium_upper_right     out = 2E(16)
    ld_medium_lower_left      out = 2F(16)
    ld_medium_lower_right     out = 30(16)
    ld_medium_up_t            out = 31(16)
    ld_medium_down_t          out = 32(16)
    ld_medium_left_t          out = 33(16)
    ld_medium_right_t         out = 34(16)
    ld_medium_cross           out = 35(16)
```

```
"                                                                              "
"        Bold Line Drawing.                                                    "
"                                                                              "
    ld_bold_begin            out = start_inverse
    ld_bold_end              out = stop_inverse
    ld_bold_horizontal       out = (' ')
    ld_bold_vertical         out = (' ')
    ld_bold_upper_left       out = (' ')
    ld_bold_upper_right      out = (' ')
    ld_bold_lower_left       out = (' ')
    ld_bold_lower_right      out = (' ')
    ld_bold_up_t             out = (' ')
    ld_bold_down_t           out = (' ')
    ld_bold_left_t           out = (' ')
    ld_bold_right_t          out = (' ')
    ld_bold_cross            out = (' ')
"                                                                              "
"   DEFAULT KEY DEFINITIONS FOR THE FULL SCREEN EDITOR                         "
"                                                                              "
"        Here is where the default function key sequences that will be        "
"   used by the full screen editor are defined.  Using the variables          "
"   defined earlier (see VARIABLES FOR FULL SCREEN EDITOR FUNCTION KEY        "
"   DEFINITONS around line fifty) the six function keys our example term-     "
"   inal has are defined.                                                      "
"        The keyword here is APPLICATION STRING (the ... indicates a          "
"   line continuation to TDU) and the name used is FSEKEYS which will         "
"   be recognized by FSE.  The out sequence is just the previously de-        "
"   fined variable strings seperated by semi-colons to make a correct         "
"   FSE command.  In addition to default function key sequences here          "
"   is a good place to put a SET TAB command if your terminal has pre-        "
"   defined hardware tabs.  Simply define a variable as was done with         "
"   k1 through k6 as  s1 = ('st 7 11 14 24 34 44 54 64') and include it       "
"   in one of the out sequences below.                                         "
"                                                                              "
    application_string...
    name = ('FSEKEYS')...
    out  = (k1 ';' k2 ';' k3 ';' k4 ';' k5 ';' k6 ';' k7 ';' k8)
    application_string...
    name = ('FSEKEYS')...
    out  = (k9 ';' k10 ';' k11 ';' k12 ';' k13 ';' k14 ';' k15)
"                                                                              "
"        Now that you have completed your TDUIN file you need to execute      "
"   the TDU command.  It should compile this file and produce a local file    "
"   called TERMLIB (or add the capsule for this terminal to a file called     "
"   TERMLIB, such as the one from UN=LIBRARY, that is alreday local).  Re-    "
"   place this file and then whenever the SCREEN,model_name command is ex-    "
"   ecuted you will see a local file called ZZZZTRM that will allow you       "
"   to interact with all NOS full screen products.                            "
"                                                                              "
"   END OF TERMINAL DEFINITION FILE FOR 721 TERMINAL                           "
```

CODE SET CONVERSION A

The code conversion information in this appendix is provided to help you interpret information coded in 6/12-bit display code or 7-bit† ASCII code when it is displayed in 6-bit display code form. The left side of table A-1 lists the 128-character ASCII character set with the corresponding 6-bit display code values. The right side of the table shows the 6/12-bit display code and 7-bit ASCII code characters as they appear when displayed in 6-bit display code format.

Table A-1. Code Conversion Chart (Sheet 1 of 4)

ASCII (128-character)			6-Bit Display Code		6/12-Bit Display Code		7-Bit ASCII Code
Character	Octal	Hexadecimal	Character	Octal	Character	Octal	Character
NUL	000	00			^5	7640	5:
SOH	001	01			^6	7641	:A
STX	002	02			^7	7642	:B
ETX	003	03			^8	7643	:C
EOT	004	04			^9	7644	:D
ENQ	005	05			^+	7645	:E
ACK	006	06			^-	7646	:F
BEL	007	07			^*	7647	:G
BS	010	08			^/	7650	:H
HT	011	09			^(	7651	:I
LF	012	0A			^)	7652	:J
VT	013	0B			^$	7653	:K
FF	014	0C			^=	7654	:L
CR	015	0D			^sp	7655	:M
SO	016	0E			^,	7656	:N
SI	017	0F			^.	7657	:O
DLE	020	10			^#	7660	:P
DC1	021	11			^[	7661	:Q
DC2	022	12			^]	7662	:R
DC3	023	13			^%	7663	:S
DC4	024	14			^"	7664	:T
NAK	025	15			^	7665	:U
SYN	026	16			^!	7666	:V
ETB	027	17			^&	7667	:W

Note: sp represents a space.

† 7-bit ASCII characters occupy the rightmost 7 bits of a 12-bit field. The leftmost 5 bits are unused.

Table A-1. Code Conversion Chart (Sheet 2 of 4)

ASCII (128-character)			6-Bit Display Code		6/12-Bit Display Code		7-Bit ASCII Code Character
Character	Octal	Hexadecimal	Character	Octal	Character	Octal	
CAN	030	18			^′	7670	:X
EM	031	19			^?	7671	:Y
SUB	032	1A			^<	7672	:Z
ESC	033	1B			^>	7673	:0
FS	034	1C			^@	7674	:1
GS	035	1D			^\	7675	:2
RS	036	1E			^^	7676	:3
US	037	1F			^;	7677	:4
sp	040	20	sp	55	sp	55	:5
! Exclamation Point	041	21	!	66	!	66	:6
" Quotation Marks	042	22	"	64	"	64	:7
# Number Sign	043	23	#	60	#	60	:8
$ Dollar Sign	044	24	$	53	$	53	:9
% Percent Sign	045	25	%	63	%	63	:+
& Ampersand	046	26	&	67	&	67	:-
′ Apostrophe	047	27	′	70	′	70	:*
(Opening Parenthesis	050	28	(	51	(	51	:/
) Closing Parenthesis	051	29	)	52	)	52	:(
* Asterisk	052	2A	*	47	*	47	:)
+ Plus	053	2B	+	45	+	45	:$
, Comma	054	2C	,	56	,	56	:=
- Dash	055	2D	-	46	-	46	:sp
. Period	056	2E	.	57	.	57	:.
/ Slant	057	2F	/	50	/	50	:/
0	060	30	0	33	0	33	:#
1	061	31	1	34	1	34	:[
2	062	32	2	35	2	35	:]
3	063	33	3	36	3	36	:%
4	064	34	4	37	4	37	:"
5	065	35	5	40	5	40	:_
6	066	36	6	41	6	41	:$\overline{!}$
7	067	37	7	42	7	42	:&
8	070	38	8	43	8	43	:′
9	071	39	9	44	9	44	:?
: Colon	072	3A	:	00	@D	7404	:<
; Semicolon	073	3B	;	77	;	77	:>
< Less than	074	3C	<	72	<	72	:@
= Equals	075	3D	=	54	=	54	:\
> Greater than	076	3E	>	73	>	73	:^
? Question Mark	077	3F	?	71	?	71	:;

Note: sp represents a space.

Table A-1. Code Conversion Chart (Sheet 3 of 4)

ASCII (128-character)			6-Bit Display Code		6/12-Bit Display Code		7-Bit ASCII Code Character
Character	Octal	Hexadecimal	Character	Octal	Character	Octal	
@ Commercial At	100	40	@	74	@A	7401	A:
A	101	41	A	01	A	01	AA
B	102	42	B	02	B	02	AB
C	103	43	C	03	C	03	AC
D	104	44	D	04	D	04	AD
E	105	45	E	05	E	05	AE
F	106	46	F	06	F	06	AF
G	107	47	G	07	G	07	AG
H	110	48	H	10	H	10	AH
I	111	49	I	11	I	11	AI
J	112	4A	J	12	J	12	AJ
K	113	4B	K	13	K	13	AK
L	114	4C	L	14	L	14	AL
M	115	4D	M	15	M	15	AM
N	116	4E	N	16	N	16	AN
O	117	4F	O	17	O	17	AO
P	120	50	P	20	P	20	AP
Q	121	51	Q	21	Q	21	AQ
R	122	52	R	22	R	22	AR
S	123	53	S	23	S	23	AS
T	124	54	T	24	T	24	AT
U	125	55	U	25	U	25	AU
V	126	56	V	26	V	26	AV
W	127	57	W	27	W	27	AW
X	130	58	X	30	X	30	AX
Y	131	59	Y	31	Y	31	AY
Z	132	5A	Z	32	Z	32	AZ
[Opening Bracket	133	5B	[	61	[	61	A0
\ Reverse Slant	134	5C	\	75	\	75	A1
] Closing Bracket	135	5D	]	62	]	62	A2
^ Circumflex	136	5E	^	76	@B	7402	A3
_ Underline	137	5F	_	65	_	65	A4

Table A-1. Code Conversion Chart (Sheet 4 of 4)

ASCII (128-character)			6-Bit Display Code		6/12-Bit Display Code		7-Bit ASCII Code Character
Character	Octal	Hexadecimal	Character	Octal	Character	Octal	
` Grave Accent	140	60	@	74	@G	7407	A5
a	141	61			^A	7601	A6
b	142	62			^B	7602	A7
c	143	63			^C	7603	A8
d	144	64			^D	7604	A9
e	145	65			^E	7605	A+
f	146	66			^F	7606	A-
g	147	67			^G	7607	A*
h	150	68			^H	7610	A/
i	151	69			^I	7611	A(
j	152	6A			^J	7612	A)
k	153	6B			^K	7613	A$
l	154	6C			^L	7614	A=
m	155	6D			^M	7615	ASP
n	156	6E			^N	7616	A,
o	157	6F			^O	7617	A.
p	160	70			^P	7620	A#
q	161	71			^Q	7621	A[
r	162	72			^R	7622	A]
s	163	73			^S	7623	A%
t	164	74			^T	7624	A"
u	165	75			^U	7625	A_
v	166	76			^V	7626	A!
w	167	77			^W	7627	A&
x	170	78			^X	7630	A´
y	171	79			^Y	7631	A?
z	172	7A			^Z	7632	A<
{ Opening Brace	173	7B	[	61	^0	7633	A>
| Vertical Line	174	7C	`	75	^1	7634	A@
} Closing Brace	175	7D	]	62	^2	7635	A\
~ Tilde	176	7E	^	76	^3	7636	A^
DEL	177	7F			^4	7637	A;

DIAGNOSTIC MESSAGES B

This appendix describes the error messages generated by NOS screen formatting. Screen formatting error messages are of four types:

- PDU syntax error messages
- PDU summary error messages
- Program dayfile error messages
- TDU syntax error messages

PDU error messages are returned as a result of an unsuccessful attempt to compile a panel using the PDU command. All PDU error messages are listed in the PDU command output file. If a PDU command is included in a batch job, the PDU summary error messages also appear in the job´s dayfile. Program dayfile messages indicate execution errors that occur during an attempt to run a program that calls screen formatting object routines.

TDU error messages are returned as a result of an unsuccessful attempt to compile a terminal definition file using the TDU command. All TDU error messages are listed in the TDU command output file. If a TDU command is included in a batch job, the TDU summary error messages also appear in the job´s dayfile.

PDU SYNTAX ERROR MESSAGES

PDU syntax error messages detect syntax errors (such as a variable name omitted on a VAR statement) encountered while scanning a panel definition file. Individual error messages begin with the characters *ERROR* and are displayed in the PDU output file as shown in the following example:

```
VAR
    !
*ERROR* EXPECTING VAR NAME AFTER VAR
```

The PDU output line in error is followed by a line containing an exclamation point that points to the position where the error occurred. The second line following the output line contains the PDU individual error message.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
ERROR EXPECTING ATTR, BOX, KEY, PANEL, TABLE, VAR or)	Unknown keyword was encountered when the beginning of a new declaration statement or the end of declarations was expected.	Correct declarations and resubmit.	PDU
ERROR EXPECTING CONSTANT AFTER =	VAR default value must be a constant.	Correct declarations and resubmit.	PDU
ERROR EXPECTING CONSTANTS AFTER RANGE	The RANGE parameter value must be two constants enclosed in parentheses.	Correct declarations and resubmit.	PDU
ERROR EXPECTING LIST AFTER MATCH	The MATCH parameter value must be enclosed in parentheses.	Correct declarations and resubmit.	PDU
ERROR EXPECTING NAME= OR ROWS= AFTER TABLE	Unknown keyword in the TABLE statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING NORMAL=(keys) OR ABNORMAL=(keys) AFTER KEY	Unknown keyword in the KEY statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING NORMAL= OR ABNORMAL= AFTER KEY	Unknown keyword in the KEY statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING PANEL NAME AFTER PANEL	A panel name is required in the PANEL statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING PHYSICAL ATTRIBUTE	A logical attribute (for example, TITLE) was specified where a physical attribute (for example, INVERSE) was expected.	Correct declarations and resubmit.	PDU
ERROR EXPECTING PRIMARY OR OVERLAY	PRIMARY or OVERLAY are the only valid panel types.	Correct declarations and resubmit.	PDU
ERROR EXPECTING QUOTED DELIMITERS	Attribute delimiters must be specified as two characters enclosed in apostrophes; for example, ATTR '()'.	Correct declarations and resubmit.	PDU
ERROR EXPECTING STRING AFTER HELP	The HELP parameter value must be a character string enclosed in apostrophes; for example, HELP='Helpful message'.	Correct declarations and resubmit.	PDU
ERROR EXPECTING TABLE DIMENSION	A table dimension (number of time the VARs are to be repeated) must be specified in a TABLE statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING TABLE NAME	A table name must be specified in a TABLE statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING TERMINATOR CHARACTER	A terminator character must be specified in a BOX statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING TERMINATOR= OR WEIGHT= AFTER BOX	Unknown keyword in the BOX statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING TYPE= AFTER PANEL	Unknown keyword in the PANEL statement.	Correct declarations and resubmit.	PDU
ERROR EXPECTING VAR NAME AFTER VAR	Each variable field declared in a VAR statement must be named.	Correct declarations and resubmit.	PDU
ERROR EXPECTING X, A, 9, N, E, YMD, MDY, DMY OR $ FORMAT	An incorrect value was specified for the FORMAT parameter.	Correct declarations and resubmit.	PDU
ERROR FIELD DECLARED DIFFERENT SIZE	A variable field in the panel image has a different length (number of underlined characters) than declared in the corresponding VAR statement.	Correct declarations and resubmit.	PDU
ERROR MORE THAN 256 BOX ELEMENTS	Only 256 lines and corners are allowed for all box figures in a single panel.	Reconstruct box figures to conform to limits.	PDU
ERROR MORE THAN 256 VARIABLES	Only 256 variable fields are allowed per panel.	Correct declarations and resubmit.	PDU
ERROR MORE THAN 32 ATTRIBUTES	Only 32 unique attribute combinations are allowed per panel.	Correct declarations and resubmit.	PDU
ERROR MORE THAN 32 KEYS	Only 32 function keys may be defined in each panel.	Correct declarations and resubmit.	PDU
ERROR MORE THAN 8 BOXES	Only eight BOX statements are allowed per panel. There may, however, be any number of individual box figures on the screen, subject to the 256 line and corner limit.	Correct declarations and resubmit.	PDU

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
ERROR MORE THAN 8 TABLES	Only eight TABLE statements are allowed per panel.	Correct declarations and resubmit.	PDU
ERROR NOT IN TABLE	A TABLEND statement was encountered without a preceding TABLE statement.	Correct declarations and resubmit.	PDU
ERROR PANEL IMAGE EXCEEDS 64 LINES	A panel may have a maximum of 64 lines.	Correct declarations and resubmit.	PDU
ERROR RANGE LOW GT HIGH	The constants in a RANGE parameter must have ascending values; for example, the second must be larger than the first.	Correct declarations and resubmit.	PDU
ERROR RANGE OR CHAR NOT ALLOWED	A CHAR type variable (the default type) cannot have a RANGE parameter.	Correct declarations and resubmit.	PDU
ERROR REAL CONSTANT FORMAT	A constant for a VAR of type REAL must be in the following format: sn.nEsm — where s is a sign (either + or -), n is one or more digits, E is the letter 'e', and m is a 1- to 3-digit number.	Correct declaration and resubmit.	PDU
ERROR SHIFT NOT ALLOWED	SHIFT cannot be specified for the CDC standard keys like BACK. Only the application keys, like F1, can be shifted.	Correct declaration and resubmit.	PDU
ERROR STRING LENGTH	A single character string within apostrophes exceeds 256 characters.	Correct declaration and resubmit.	PDU
ERROR TABLE DIMENSION REQUIRED	The number of rows in a TABLE must be declared for each table. There is no default.	Correct declaration and resubmit.	PDU
ERROR TABLE NAME REQUIRED	Tables must have a name specified in the TABLE statement.	Correct declaration and resubmit.	PDU
ERROR TABLE PARAMETER	More than two parameters were specified in the TABLE statement.	Correct declaration and resubmit.	PDU
ERROR TERMINATOR CHAR REQUIRED	A terminator character enclosed in apostrophes must be specified for each BOX statement.	Correct declaration and resubmit.	PDU
ERROR TOO MANY ATTR PARAMETERS	More positional parameters than allowed were specified in the ATTR statement.	Correct declaration and resubmit.	PDU
ERROR TOO MANY VAR PARAMETERS	More positional parameters than allowed were specified in the VAR statement.	Correct declaration and resubmit.	PDU
ERROR TWO VAR NAMES	More than one name was specified in a single VAR statement. A single VAR can have only one name.	Correct declaration and resubmit.	PDU
ERROR TYPE/FORMAT MISMATCH IN PRECEDING VAR	The format specified in the VAR statement is not compatible with the data type.	Correct declaration and resubmit.	PDU
ERROR UNEXPECTED END OF FILE	The end of the panel definition file was encountered before the end of declarations; for example, before the terminating).	Correct declaration and resubmit.	PDU
ERROR UNKNOWN KEYWORD	The keyword specified is not allowed for this statement.	Correct declaration and resubmit.	PDU
ERROR UNTERMINATED STRING	A string with no closing apostrophe was encountered.	Correct declaration and resubmit.	PDU
ERROR VALIDATION TABLE OVERFLOW	The panel contains too much variable-related information. The validation table is an internal table used to store all validation and help information for the panel. It can contain approximately 4000 characters.	Simplify the panel by reducing the number of variable fields, or by reducing the amount of validation and/or help information specified for panel variable fields.	PDU
ERROR VALUE TYPE MISMATCH	The initial VALUE specified in a VAR statement is not the same type as the declared TYPE; for example, an integer initial value for a CHAR type variable.	Correct declaration and resubmit.	PDU
ERROR VAR DECLARED TWICE	Two VAR statements using the same variable name were encountered. Variable names must be unique.	Correct declarations and resubmit.	PDU
ERROR VAR NAME NOT SPECIFIED	Each VAR statement must specify a variable name.	Correct declarations and resubmit.	PDU

PDU SUMMARY ERROR MESSAGES

PDU summary error messages indicate the type of error that caused the compilation to fail. Summary error messages begin with the characters PANEL- and are listed at the end of the PDU output file. If the PDU command is included in a batch job, the summary error messages also appear in the job´s dayfile.

Syntax errors in the panel definition file generate both PDU individual and summary error messages. All other panel definition errors produce only a summary error message.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
PANEL - ERROR IN xxxxxx CAN'T OPEN FILE yyyyyyy	The specified file containing the panel definitions could not be opened; for example, was not a local file.	Correct file name or attach, get, or create the definition file.	PDU
PANEL - ERROR IN xxxxxx DECLARATIONS	Preceding errors in the declaration part of the panel definition caused compilation to fail. The image is not scanned.	Correct errors and resubmit.	PDU
PANEL - ERROR IN xxxxxx END OF FILE DURING DEFINITIONS	The end of the panel definition file was encountered before the end of declarations; for example, before the terminating).	Supply missing) or otherwise correct the panel definition and resubmit.	PDU
PANEL - ERROR IN xxxxxx NO DEFINITION ON IMAGE	An empty panel definition file was submitted. The definition file must have at least one line.	Correct definition and resubmit.	PDU
PANEL - ERROR IN xxxxxx SCREEN IMAGE	A previously noted error in the panel image caused compilation to fail.	Correct image and resubmit.	PDU
PANEL - ERROR IN xxxxxx UNRECOGNIZED PARAMETER yyy	An unrecognized (probably misspelled) parameter keyword was specified on the PANEL statement.	Correct parameter specifications and resubmit.	PDU

PROGRAM DAYFILE ERROR MESSAGES

As the name implies, program dayfile error messages are listed in the dayfile of the job that initiated execution of the program. Program dayfile messages begin with the name of the screen formatting object routine that encountered the error.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
SFCLOS PANEL xxxxxxx ALREADY CLOSED	An attempt was made to close a panel more than once.	Check program logic for a redundant SFCLOS subroutine call for panel xxxxxxx.	SFCLOS
SFCLOS PANEL xxxxxxx NOT IN PLT	An attempt was made to close a panel that was never opened.	Check program logic for a missing SFOPEN subroutine call for panel xxxxxxx.	SFCLOS
SFCLOS PANEL xxxxxxx NOT OPENED	An attempt was made to show a panel that was never opened.	Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSSNO subroutine call.	SFSSMO
SFCLOS PANEL xxxxxxx NOT UNLOADED	The fast dynamic loader was unable to unload panel xxxxxxx.	Call site analyst.	SFCLOS
SFOPEN PANEL xxxxxxx BAD ENTRY FORMAT	The passloc/entry list in routine LCP is incorrect.	Call site analyst.	SFOPEN
SFOPEN PANEL xxxxxxx BAD GROUP NAME	The group name of the panel library being used is incorrect.	Call site analyst.	SFOPEN
SFOPEN PANEL xxxxxxx BAD LIBRARY LIST	The library list in routine LCP is incorrect.	Call site analyst.	SFOPEN
SFSREA PANEL xxxxxxx NOT OPENED	An attempt was made to show a panel that was never opened.	Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSREA subroutine call.	SFSSMO
SFSWRI PANEL xxxxxxx NOT OPENED	An attempt was made to show a panel that was never opened.	Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSWRI subroutine call.	SFSWRI
SFSWRI PANEL xxxxxxx NOT PRIMARY	An attempt was made to write an overlay panel before any primary panel was written (for example, while the screen display is still in line mode).	Check program logic to ensure that the primary panel is written on the screen before overlay panels are called.	SFSWRI

TDU SYNTAX ERROR MESSAGES

TDU syntax error messages detect syntax errors encountered while scanning a terminal definition file. The TDU messages are prefixed with the line:

```
    TDU TERMINATED WITH ERRORS
```

Syntax error messages are displayed in the TDU output file (which is an ASCII file) as shown in the following example:

```
    INVALID COMMUNICATIONS TYPE
    communications  type = bisynch
                                 !
```

The first line contains the TDU syntax error message. The second line is the line of the terminal definition file in error, followed by a line with an exclamation point that points to the position where the error occurred.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
CONTINUATION EXCEEDS 256 CHARACTERS	The total number of characters in a line and its continuation exceeds 256.	Reformat using variables and minimize indentation.	TDU
CURSOR_BIAS OUT OF RANGE, MUST BE -255 TO 255	Cursor_bias must be within range -255 <= cursor_bias <= 255.	Correct bias and resubmit.	TDU
DEFINITION FILE NOT FOUND	The user returned the file ZZZZTRM which containes the terminal definitions.	Re-issue the SCREEN or LINE command.	TDU
DOUBLY DEFINED PARAMETER xxxxxx	A parameter appeared twice in the same statement.	Check mixed usage of keyword parameters.	TDU
DUPLICATE PARAMETERS, BOTH "IN" AND "INOUT"	You specified both parameters in the same statement.	Possible confusion when using parameters. Use "IN" and "OUT" only when the character sequences differ, otherwise use "INOUT".	TDU
DUPLICATE PARAMETERS, BOTH "OUT" AND "INOUT"	Both parameters were specified in the same statement with an Input/output verb.	Possible confusion when using parameters. Use "IN" and "OUT" only when the character sequences differ, otherwise use "INOUT".	TDU
EMPTY INPUT FILE	The TDU input file contained only blank lines (or no lines).	Check input file.	TDU
EXPECTING xxxxxx	TDU was expecting to find the indicated symbol, but did not.	Correct statement and resubmit.	TDU
EXPECTING VERB OR VARIABLE, FOUND xxxxxx	A statement began with a symbol other than a name, such as an integer, boolen, string, and so on.	Correct statement and resubmit.	TDU
INTEGER OVERFLOW xxxxxx	Specified integer exceeds CDC integer size.	Correct integer value and resubmit.	TDU
INTEGER TOO LARGE xxxxxx	Specified integer exceeds CDC integer ceiling.	Correct integer value and resubmit.	TDU
INVALID xxxxxx	The indicated symbol is not allowed at the location where it was found.	Correct statement and resubmit.	TDU
INVALID CURSOR_ENCODING	Communications value not from allowed set. Must be ASYNCH, SYNCH, or SNA.	Correct the communication value.	TDU
INVALID "MOVE_PAST.." OR "CHAR_PAST.." TYPE	An incorrect value was assigned to the TYPE parameter for one of the verbs MOVE_PAST_SIDE, MOVE_PAST_TOP, MOVE_PAST_BOTTOM, CHAR_PAST_SIDE, or CHAR_PAST_LAST_POSITION.	Use only STOP_NEXT, SCROLL_NEXT, HOME_NEXT, WRAP_ADJACENT_NEXT, or WRAP_SAME_NEXT for the value and resubmit.	TDU
INVALID NAME—MAY ONLY BE ALPHABETIC AND NUMERIC CHARACTERS	The value assigned to the VALUE parameter of the MODEL_NAME statement used a character other than A-Z or 0-9.	Use only alphabetic and numeric characters in the name.	TDU
INVALID TYPE-ONLY STRING, INTEGER, OR VARIABLE ALLOWED	A boolean, undeclared variable, or other symbol was encountered in a character string sequence.	Check for a misspelled name, missing apostrophe, and so on.	TDU
INVALID VERB OR MISSING "=" IN VARIABLE ASSIGNMENT	A statement began with a name which TDU did not recognize so it assumed statement was a variable declaration; but there was no "=" symbol.	Check for misspelled statement or missing "=" symbol.	TDU
ITEM xxxxxx IS SUPERSET OF A PREVIOUS ITEM	The leading characters of the input character sequence are the same as an entire character sequence encountered earlier.	All input character sequences must be unique.	TDU
NAME IS REQUIRED	The MODEL_NAME statement was missing or the name was invalid.	You must give your terminal definition file a unique name.	TDU
NAME MUST BE 1 TO 6 CHARACTERS	The value assigned to the VALUE parameter of the MODEL_NAME statement used 0 or more than 6 characters.	Use at least 1 character but no more than 6 characters in the name.	TDU

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
NO ROOM IN TABLE FOR xxxxxx	TDU internal tables exceeded available storage.	Increase the job's field length limit and retry.	TDU
NOT YET IMPLEMENTED xxxxxx	Reserved for future implementation.	None.	TDU
NUMBER OF COLUMNS MUST RANGE FROM 0 to 511	You specified too large a number of columns. It should be within the range 0 to 511.	Correct number of columns and resubmit.	TDU
NUMBER OF ROWS MUST RANGE FROM 0 TO 63	You specified too large a number of rows. It chould be within the range 0 to 63.	Correct number of rows and resubmit.	TDU
"OUT" REQUIRED FOR SET_SIZE	The SET_SIZE statement was used without specifying the OUT parameter.	For every screen size you must specify the character sequence that switches the terminal into that size.	TDU
REQUIRED PARAMETER MISSING xxxxxx	The indicated parameter must be specified when this verb is used.	Supply necessary parameter and resubmit.	TDU
STRING OVERFLOW xxxxxx	The total number of characters in a string exceeds 256.	See initialization verb section. If string is part of a terminal function other than initialization, look for a way to shorten it.	TDU
TABLE OVERFLOW xxxxxx	TDU internal tables exceeded available storage.	Increase the job's field length limit and retry.	TDU
TABLE OVERFLOW DURING OPTIMIZATION	TDU internal tables exceeded availaable storage.	Increase the job's field length limit and retry.	TDU
TDU TERMINATED WITH ERRORS	TDU encountered errors in the terminal definition file as indicated by other messages.	Correct errors in the TDU input file.	TDU
TERMINAL DEFINITION NOT FOUND	There is no file named TERMLIB which contains the specified terminal definitions.	None.	TDU
TERMINAL MODEL NOT YET SPECIFIED	The user has not previously issued a SCREEN or LINE command and so must specify the terminal model name.	Specify the terminal model name so the SCREEN and LINE commands can be issued without specifying the terminal model.	TDU
TOO MANY xxxxxx	A value list was used with a parameter which only allows a single value.	Correct statement and resubmit.	TDU
TOO MANY SCREEN SIZES SPECIFIED, MAXIMUM 4	You specified too many screen sizes.	Choose your four favorite screen sizes.	TDU
UNBALANCED xxxxxx	The indicated symbol should be used in pairs. It was not.	Check for a missing parenthesis or apostrophe.	TDU
UNEXPECTED xxxxxx	TDU did not expect to fine the indicated symbol where it did.	Correct statement and resubmit.	TDU
UNKNOWN KEYWORD xxxxxx	TDU did not recognize a parameter.	Check for misspelling, or extra parenthesis or apostrophe.	TDU
VALUE RANGE NOT ALLOWED xxxxxx	TDU does not use value ranges.	Use a value list.	TDU
VARIABLE xxxxxx HAS NOT BEEN DECLARED	The indicated variable was not previously defined.	Check for misspelling or missing apostrophe.	TDU
VERB xxxxxx APPEARS TWICE	Input, Output, and Input/output statements may only appear once.	Delete the redundant statement.	TDU

GLOSSARY C

Alternate Intensity Character Display

A CRT display characteristic in which certain characters or character strings are highlighted by displaying them at a different light intensity than the surrounding text.

Application Program

A program resident in a host computer that uses the Network Access Method and provides an information storage, retrieval, and/or processing communication network.

ASCII

American National Standard Code for Information Interchange.

CDC Standard Function Keys

The following function keys are defined as CDC standard function keys: NEXT, HELP, BACK, STOP, FWD, BKW, UP, and DOWN.

COBOL

Common Business Oriented Language. This higher-level language simplifies the programming of business data applications.

Declaration Section

In NOS screen formatting, the part of a panel definition file that defines the display characteristics and data type characteristics of information appearing in a panel.

Direct Access File

A type of NOS file which allows you to make editing changes directly on the permanent copy of the file. Contrast with Indirect Access File.

Editing Function Keys

The following terminal keys are defined as editing function keys: INSERT (character/line), DELETE (character/line), ERASE,TAB (forward), TAB (backward), CLEAR (page/end of line).

FORTRAN

Formula Translation. A language that solves algebraic and scientific problems using symbols and statements that closely resemble mathematical notation.

Full Screen Editor (FSE)

A NOS text editor which allows you to edit files in either line mode or screen mode.

Function Key

Any of a number of special keys (apart from the standard typewriter keys) on a user terminal which are used to request a specific action by the application program. The number and type of functions keys available on a keyboard differs depending on the terminal model. See also CDC Standard Function Keys, Editing Function Keys, and Programmable Function Keys.

Image Section

In NOS screen formatting, the part of a panel definition file in which you define the format or layout of a panel.

Indirect Access File

A type of NOS file which allows you to make editing changes on a local copy of the file without affecting the permanent copy of the file. When you are finished editing the local copy, you can either replace the permanent copy with the local (edited) copy or else discard the local copy. Contrast with Direct Access File.

Inverse Video Display

A CRT display characteristic in which characters or character strings are highlighted by displaying the characters darkened against a lighted background, rather than vice versa.

Line Mode

A method of interactive job entry in which job statements or commands are entered and executed on a line by line basis. Contrast with Screen Mode.

NOS Procedure

A series of NOS commands that resides in a separate file or file record and that is structured to perform a specific subroutine-like function. NOS procedures can be called from an executing job or from another procedure.

Object Routine

A section of program code which resides on a common file or library and which performs a specific, frequently repeated function. An object routine can be loaded and called as a subroutine by an executing application program.

Panel

In NOS screen formatting, a formatted screen defined using the Panel Definition Utility (PDU). An application program uses a panel to display data or request user input at the terminal.

Panel Definition File

In NOS screen formatting, a NOS text file which defines a panel format. The panel definition file must be compiled and stored in a user library before it can be called by an executing application program.

Panel Definition Utility (PDU)

In NOS screen formatting, the utility used to create and maintain panels and panel libraries.

Pascal

A general usage high-level programming language.

Programmable Function Keys

The numbered function keys on a user terminal. The programmable function keys are usually labelled F1,F2,...,Fn or PF1,PF2,...,PFn.

Screen Mode

A method of interactive job entry in which formatted display screens are used to display output information or to request user input of job parameters or program data. Contrast with Line Mode.

Terminal Definition Utility (TDU)

In NOS screen formatting, the utility used to compile definition files to be loaded defining terminal key functions.

User Library

A file of binary modules that can be used by the loader to load routines and satisfy externals. It contains tables referencing the assembled central processor programs, subroutines, text records, or overlays.

Validation Checking

The process of testing input values submitted for procedure parameters, program variables, or other types of input variables to ensure that the entered values meet any specified format or range requirements.

SAMPLE PROGRAMS D

This appendix contains a FORTRAN 5 program, a COBOL 5 program, and a Pascal program that demonstrate how panels can be used in application programs to perform program input and output operations. The panel definition files used to create each panel are also included in this appendix, so you can create panel libraries for sample programs and run them in screen mode.

NOTE

The first line of the panel definition file must always be left-justified.

FORTRAN PROGRAM ANGLE3

Figure D-1 presents the listing for a FORTRAN program called ANGLE3. ANGLE3 calculates the area of a triangle from values entered by the user. ANGLE3 uses five different panels. The panel definition files for ANGLE3 panels are presented in figures D-2 through D-6.

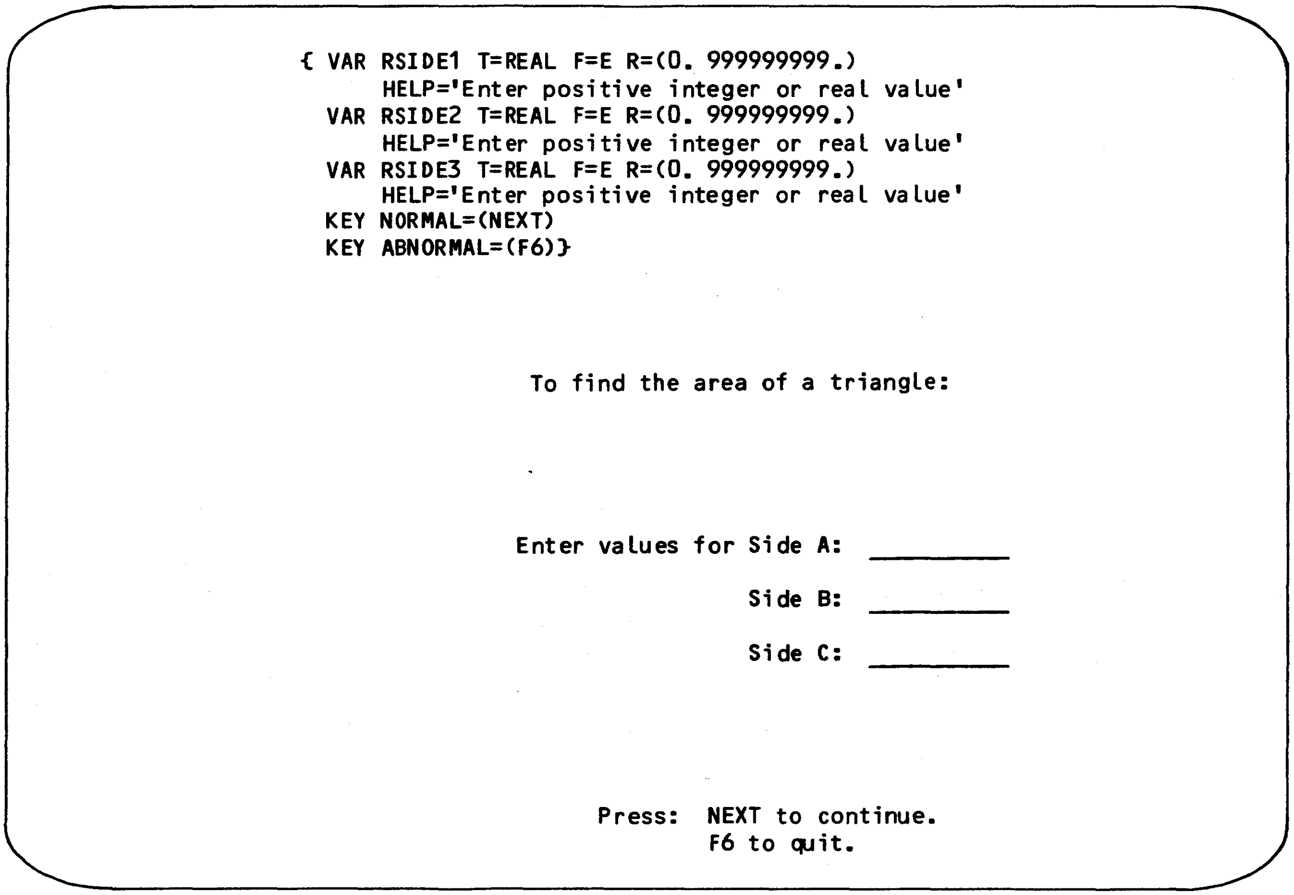
Figures D-2 and D-3 are the panel definition files for the ANGLE3 input and output panels. The input panel is called TRYIN and the output panel is called TRYOUT.

```
      PROGRAM ANGLE3
C     ***THIS PROGRAM CALCULATES THE AREA OF A TRIANGLE***
      INTEGER STAT, KTYPE, KORD, SW, F1, QUIT, NEXT, FKEY, CDCKEY
      REAL RSIDE(3), S, RDCL, AREA
      CHARACTER INPAN*30, OUTPAN*40, DUMMY*40
      CHARACTER* (*) TRYIN, TRYOUT, MSGOVL1, MSGOVL2, BLNKOVL
      PARAMETER(TRYIN='TRYIN', TRYOUT='TRYOUT', MSGOVL1='MSGOVL1',
     +          MSGOVL2='MSGOVL2', BLNKOVL='BLNKOVL')
      PARAMETER (F1=1, QUIT=6, FKEY=0, CDCKEY=1, NEXT=1)
C     ***OPEN ALL PANELS; PRINT DIAGNOSTIC MESSAGE
C        IF SFOPEN IS UNSUCCESSFUL.***
      CALL SFOPEN(TRYIN,STAT)
      IF(STAT .NE. 0) THEN
         PRINT *,'PANEL TRYIN NOT OPENED; STAT=',STAT
         STOP
      ENDIF
      CALL SFOPEN(TRYOUT,STAT)
      IF(STAT .NE. 0) THEN
         PRINT*,'PANEL TRYOUT NOT OPENED; STAT=',STAT
         STOP
      ENDIF
      CALL SFOPEN(MSGOVL1,STAT)
      IF(STAT .NE. 0) THEN
         PRINT*,'PANEL MSGOVL1 NOT OPENED; STAT=',STAT
         STOP
      ENDIF
      CALL SFOPEN(MSGOVL2,STAT)
      IF(STAT .NE. 0) THEN
         PRINT*,'PANEL MSGOVL2 NOT OPENED; STAT=',STAT
         STOP
      ENDIF
      CALL SFOPEN(BLNKOVL,STAT)
      IF(STAT .NE. 0) THEN
         PRINT*,'PANEL BLNKOVL NOT OPENED; STAT=',STAT
         STOP
       ENDIF
C     ***READ INPUT STRING (INPAN)***
20    CALL SFSREA(TRYIN,INPAN)
C     ***TEST FOR QUIT KEY; IF PRESSED, TERMINATE
C        PROGRAM.***
      CALL SFGETK(KTYPE,KORD)
      IF(KTYPE .EQ. FKEY .AND. KORD .EQ. QUIT) THEN
         CALL SFCLOS(TRYIN,1)
         STOP
      ENDIF
```

Figure D-1. FORTRAN Program ANGLE3 (Sheet 1 of 2)

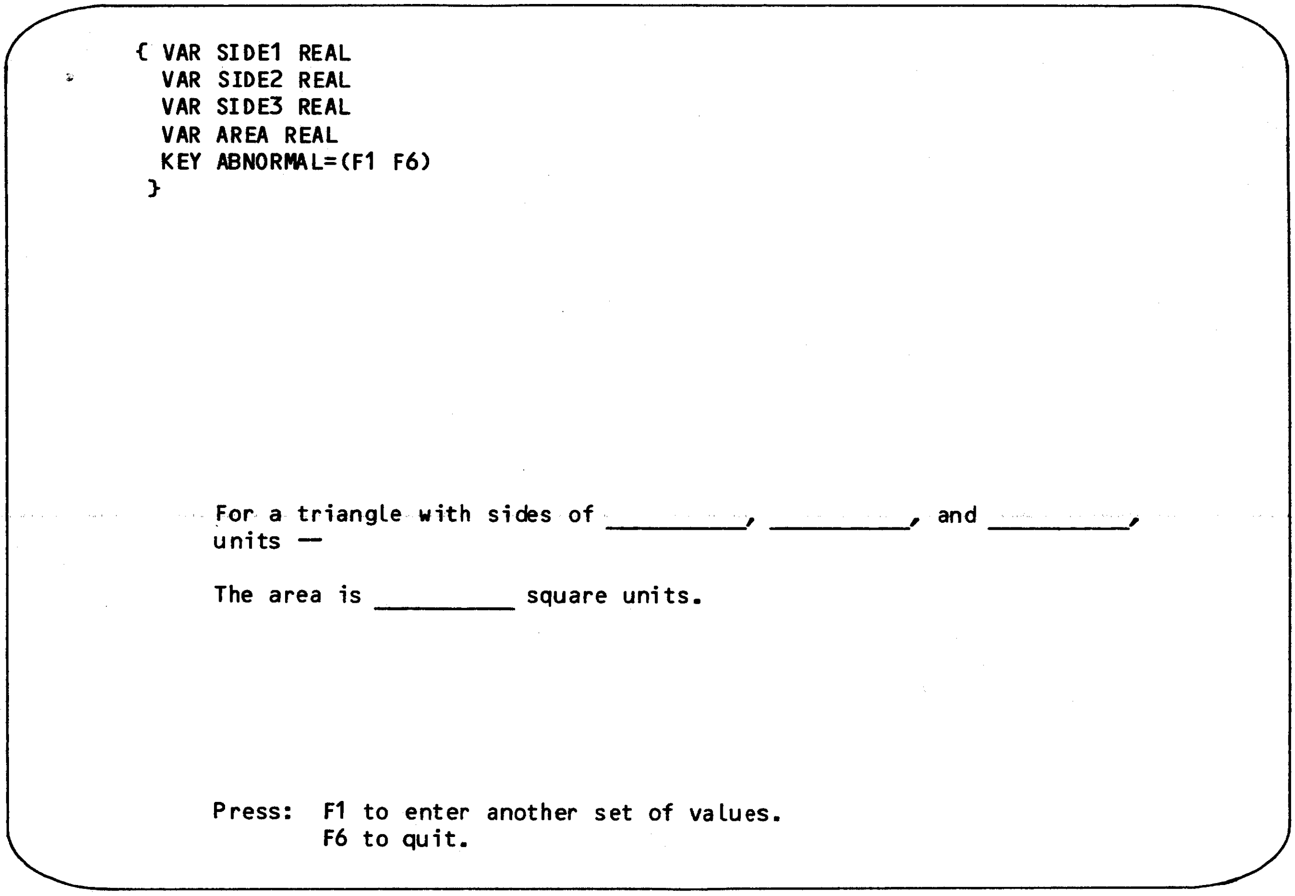
```
C     ***TEST FOR MSGOVL1 SWITCH SETTING. IF SET, CALL
C        BLNKOVL AND CLEAR SWITCH; OTHERWISE, CONTINUE.***
      IF (SW .NE. 0) THEN
         CALL SFSWRI(BLNKOVL,DUMMY)
         SW=0
      ENDIF
C     ***CONVERT INPUT TO REAL VARIABLES***
      READ(INPAN,'(3F10.0)')RSIDE
C     ***CALCULATE AREA OF TRIANGLE***
      S=(RSIDE(1)+RSIDE(2)+RSIDE(3))/2.0
      RDCL=S*(S-RSIDE(1))*(S-RSIDE(2))*(S-RSIDE(3))
      IF(RDCL.LE.0.0) THEN
C        ***IF VALUES ENTERED DO NOT FORM A VALID TRIANGLE,
C        USE MSGOVL1 TO DISPLAY DIAGNOSTIC MESSAGE.***
         CALL SFSWRI(MSGOVL1,DUMMY)
         SW=1
      ELSE
C        ***CALCULATE AREA.  IF AREA EXCEEDS MAXIMUM ALLOWED
C        VALUE (9999999.99), USE MSGOVL2 TO DISPLAY
C        DIAGNOSTIC MESSAGE.***
         AREA=SQRT(RDCL)
         IF (AREA.GT.9999999.99) THEN
            CALL SFSWRI(MSGOVL2,DUMMY)
            SW=1
         ELSE
C           ***CONVERT REAL VARIABLES TO CHARACTER VARIABLES,
C           AND PACK IN OUTPUT STRING (OUTPAN).***
            WRITE(OUTPAN,'(4F10.2)')RSIDE,AREA
C           ***CALL SFSSHO TO OUTPUT RESULTS.***
C           ***NOTE - SFSSHO, BELOW, USES A DUMMY VARIABLE FOR THE
C           INPUT STRING TO ALLOW PANEL INPUT THROUGH FUNCTION
C           KEYS.***
            CALL SFSSHO(TRYOUT,OUTPAN,DUMMY)
         ENDIF
      ENDIF
C     ***TEST FOR FUNCTION KEY PRESSED (F1 OR F6) -
C     IF F1, REDISPLAY TRYIN PANEL TO GET NEXT SET OF
C     VARIABLES.  IF F6, CLOSE TRYIN PANEL AND TERMINATE
C     PROGRAM.***
      CALL SFGETK(KTYPE,KORD)
      IF (KTYPE .EQ. CDCKEY .AND. KORD .EQ. NEXT) GO TO 20
      IF (KTYPE .EQ. FKEY .AND. KORD .EQ. F1) GO TO 20
      CALL SFCLOS(TRYIN,1)
      END
```

Figure D-1. FORTRAN Program ANGLE3 (Sheet 2 of 2)



```
{ VAR RSIDE1 T=REAL F=E R=(0. 999999999.)
       HELP='Enter positive integer or real value'
  VAR RSIDE2 T=REAL F=E R=(0. 999999999.)
       HELP='Enter positive integer or real value'
  VAR RSIDE3 T=REAL F=E R=(0. 999999999.)
       HELP='Enter positive integer or real value'
  KEY NORMAL=(NEXT)
  KEY ABNORMAL=(F6)}



                      To find the area of a triangle:




                    Enter values for Side A:  __________

                                     Side B:  __________

                                     Side C:  __________





                          Press:  NEXT to continue.
                                  F6 to quit.
```

Figure D-2. TRYIN Panel Definition File



```
{ VAR SIDE1 REAL
  VAR SIDE2 REAL
  VAR SIDE3 REAL
  VAR AREA REAL
  KEY ABNORMAL=(F1 F6)
 }









      For a triangle with sides of __________, __________, and __________,
      units —

      The area is __________ square units.






      Press:  F1 to enter another set of values.
              F6 to quit.
```

Figure D-3. TRYOUT Panel Definition File

Figures D-4 and D-5 show the panel definition files for two error message panels. These panels, named MSGOVL1 and MSGOVL2, are called by ANGLE3 in response to invalid user input. Both MSGOVL1 and MSGOVL2 are overlay panels that modify the ANGLE3 input (TRYIN) panel. When either MSGOVL1 or MSGOVL2 is called, the corresponding error message is displayed in inverse video in the upper right corner of the input panel.

Figure D-6 is the panel definition file for an overlay panel called BLNKOVL. When either of the error messages defined by MSGOVL1 or MSGOVL2 is displayed, the user can indicate his or her intention to enter new values by pressing the F1 function key. Upon detecting that F1 has been pressed, the program calls BLNKOVL to blank out the error message.

```
{ PANEL MSGOVL1 OVERLAY
  ATTR '[]' P=INVERSE
  KEY ABNORMAL=(F1 F6)}

                            [THE VALUES ENTERED DO NOT FORM A TRIANGLE]
                            [             --PLEASE REENTER            ]
```

Figure D-4. MSGOVL1 Panel Definition File

```
{ PANEL MSGOVL2 OVERLAY
  ATTR '[]' P=INVERSE
  KEY ABNORMAL=(F1 F6)}

                            [AREA EXCEEDS MAXIMUM ALLOWABLE VALUE OF  ]
                            [  9999999.99 - REENTER VALUES OR QUIT.   ]
```

Figure D-5. MSGOVL2 Panel Definition File

```
{ PANEL BLNKOVL OVERLAY
  ATTR '[]' L=TEXT
  KEY ABNORMAL=(F1 F6)}

                           [                                         ]
                           [                                         ]
```

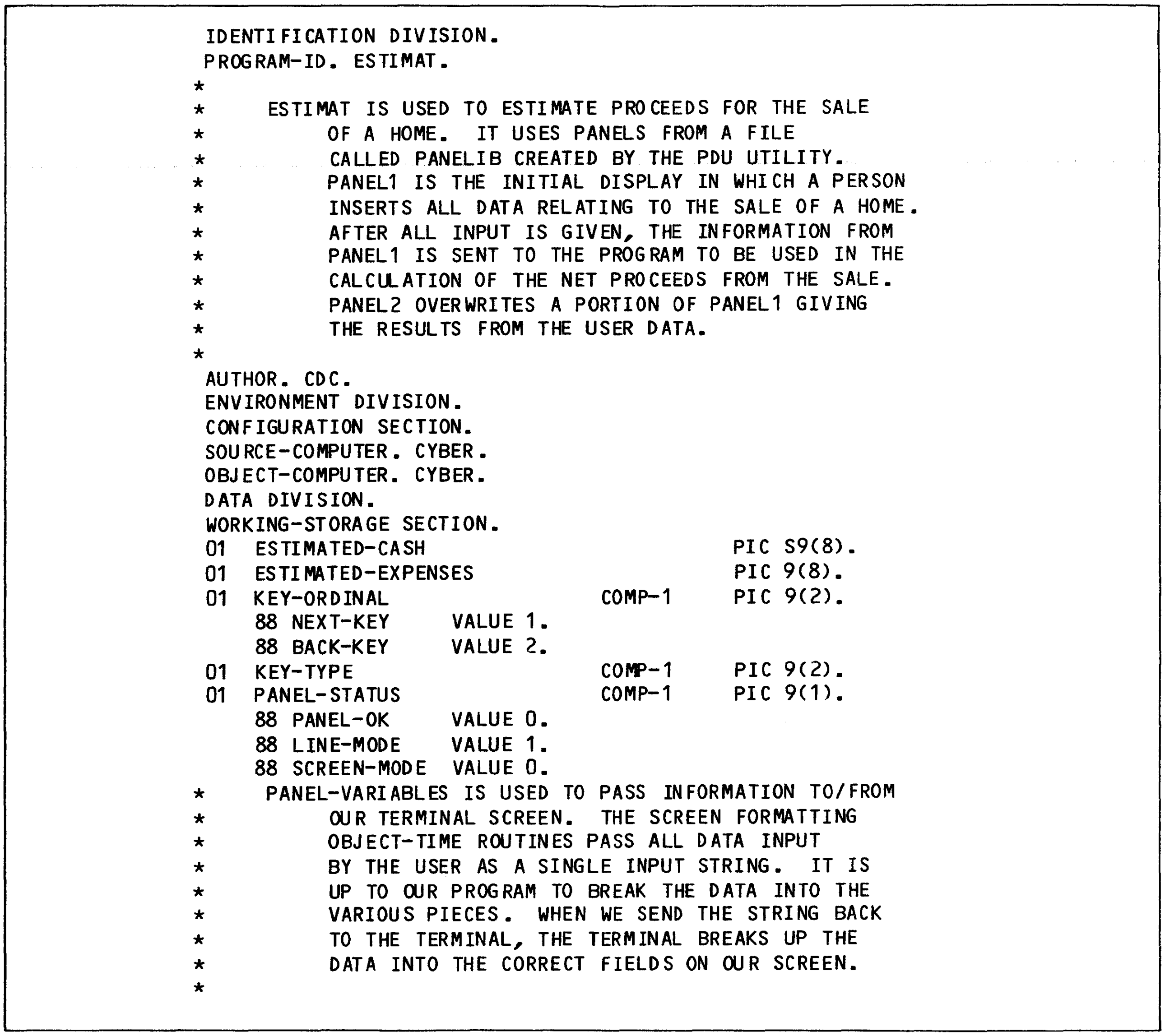
Figure D-6. BLNKOVL Panel Definition File

COBOL PROGRAM ESTIMAT

Figure D-7 is a listing of the COBOL program ESTIMAT. ESTIMAT is used to estimate the proceeds from the sale of a home. ESTIMAT uses two panels, an input panel called PANEL1 and an output panel called PANEL2. The panel definition files for PANEL1 and PANEL2 are shown in figures D-8 and D-9, respectively.

Figure D-8 shows the panel definition file for PANEL1. PANEL1 accepts and validates user input, and returns the input to the application program.

Figure D-9 is the panel definition file for PANEL2. PANEL2 adds three lines of output information to the PANEL1 screen display.



```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ESTIMAT.
*
*    ESTIMAT IS USED TO ESTIMATE PROCEEDS FOR THE SALE
*         OF A HOME.  IT USES PANELS FROM A FILE
*         CALLED PANELIB CREATED BY THE PDU UTILITY.
*         PANEL1 IS THE INITIAL DISPLAY IN WHICH A PERSON
*         INSERTS ALL DATA RELATING TO THE SALE OF A HOME.
*         AFTER ALL INPUT IS GIVEN, THE INFORMATION FROM
*         PANEL1 IS SENT TO THE PROGRAM TO BE USED IN THE
*         CALCULATION OF THE NET PROCEEDS FROM THE SALE.
*         PANEL2 OVERWRITES A PORTION OF PANEL1 GIVING
*         THE RESULTS FROM THE USER DATA.
*
 AUTHOR. CDC.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ESTIMATED-CASH                           PIC S9(8).
 01  ESTIMATED-EXPENSES                       PIC 9(8).
 01  KEY-ORDINAL                 COMP-1       PIC 9(2).
     88 NEXT-KEY     VALUE 1.
     88 BACK-KEY     VALUE 2.
 01  KEY-TYPE                    COMP-1       PIC 9(2).
 01  PANEL-STATUS                COMP-1       PIC 9(1).
     88 PANEL-OK     VALUE 0.
     88 LINE-MODE    VALUE 1.
     88 SCREEN-MODE  VALUE 0.
*    PANEL-VARIABLES IS USED TO PASS INFORMATION TO/FROM
*         OUR TERMINAL SCREEN.  THE SCREEN FORMATTING
*         OBJECT-TIME ROUTINES PASS ALL DATA INPUT
*         BY THE USER AS A SINGLE INPUT STRING.  IT IS
*         UP TO OUR PROGRAM TO BREAK THE DATA INTO THE
*         VARIOUS PIECES.  WHEN WE SEND THE STRING BACK
*         TO THE TERMINAL, THE TERMINAL BREAKS UP THE
*         DATA INTO THE CORRECT FIELDS ON OUR SCREEN.
*
```

Figure D-7. COBOL Program ESTIMAT (Sheet 1 of 5)

```
 01  PANEL-VARIABLES.
   02  PANEL1-VARIABLES.
     03  PANEL1-ALPHA-VARIABLES.
             05  PANEL1-OWNER              PIC X(26).
             05  PANEL1-DATE               PIC X(8).
             05  PANEL1-SPERSON            PIC X(26).
     03  PANEL1-NUMERIC-VARIABLES.
             05  PANEL1-SPRICE             PIC ZZZZZZZ9.
             05  PANEL1-MORTGAG            PIC ZZZZZZZ9.
             05  PANEL1-PAYCD              PIC ZZZZZZZ9.
             05  PANEL1-HOMEILN            PIC ZZZZZZZ9.
             05  PANEL1-ABSUPD             PIC 999.
             05  PANEL1-TAXES              PIC ZZZZZZZ9.
             05  PANEL1-RFEES              PIC 99.
             05  PANEL1-REPAIRS            PIC ZZZZZZZ9.
             05  PANEL1-CLOSFEE            PIC 99.
             05  PANEL1-REALFEE            PIC 9.
   02  PANEL2-VARIABLES.
             05  PANEL2-SPRICE             PIC ZZZZZZZ9.
             05  PANEL2-EXPENSE            PIC ZZZZZZZ9.
             05  PANEL2-ECASH              PIC -ZZZZZZ9.
*
*    HOLD-VARIABLES IS USED TO RETRIEVE VARIABLES
*         FROM THE PANEL IN INTEGER FORMAT.
*
 01  HOLD-VARIABLES.
     03  HOLD-SPRICE             COMP-1    PIC 9(8).
     03  HOLD-MORTGAG            COMP-1    PIC 9(8).
     03  HOLD-PAYCD              COMP-1    PIC 9(8).
     03  HOLD-HOMEILN            COMP-1    PIC 9(8).
     03  HOLD-TAXES              COMP-1    PIC 9(8).
     03  HOLD-REPAIRS            COMP-1    PIC 9(8).
     03  HOLD-CLOSFEE            COMP-1    PIC 9(2).
     03  HOLD-REALFEE            COMP-1    PIC 9(1).

 PROCEDURE DIVISION.
 START-PROGRAM.
*
*     OPEN PANELS "PANEL1" AND "PANEL2" FOR USE BY THE PROGRAM.
*
     ENTER SFOPEN USING "PANEL1", PANEL-STATUS.
     IF NOT PANEL-OK
             GO TO STOP-PROGRAM
     END-IF.
     ENTER SFOPEN USING "PANEL2", PANEL-STATUS.
     IF NOT PANEL-OK
             MOVE 1 TO PANEL-STATUS
             PERFORM CLOSE-PANELS
             GO TO STOP-PROGRAM
     END-IF.
```

Figure D-7. COBOL Program ESTIMAT (Sheet 2 of 5)

```
 DISPLAY-PANEL.
*
*     CALL TO SFSREA DISPLAYS PANEL1 AT THE TERMINAL
*          WITH THE DEFAULT VALUES.
*
*     IT ALSO CAUSES THE PROGRAM TO READ THE RESULTS
*          FROM THE USER INPUT AND PLACE THEM IN
*          PANEL1-VARIABLES.
*
     ENTER SFSREA USING "PANEL1",  PANEL1-VARIABLES.
*
*     SFGETK RETURNS THE FUNCTION KEY TYPED AT THE TERMINAL
*     (REFER TO KEY STATEMENTS IN THE PANEL DEFINITION).
*
     ENTER SFGETK USING KEY-TYPE, KEY-ORDINAL.
*
*     CHECK FOR -BACK- KEY
*
     IF BACK-KEY
         GO TO START-OVER
     END-IF.
*
*     CHECK FOR -NEXT- KEY
*
     IF NOT NEXT-KEY
             SET LINE-MODE TO TRUE
             PERFORM CLOSE-PANELS
             GO TO STOP-PROGRAM
     END-IF.
*
*     THE FOLLOWING SFGETI CALLS RETRIEVE ALL INTEGER VARIABLES
*          RIGHT JUSTIFIED SO THAT THE COBOL PROGRAM CAN USE THEM
*          IN COMPUTATIONAL STATEMENTS. IF WE USED THE
*          VARIABLES FROM PANEL1-VARIABLES, WE WOULD HAVE
*          TO "RIGHT-JUSTIFY" THEM AND "REPLACE LEADING SPACES
*          BY ZEROS" BEFORE USING THEM IN CALCULATIONS.
*
     ENTER SFGETI USING "SPRICE",  HOLD-SPRICE.
     ENTER SFGETI USING "MORTGAG", HOLD-MORTGAG.
     ENTER SFGETI USING "PAYCD",   HOLD-PAYCD.
     ENTER SFGETI USING "HOMEILN", HOLD-HOMEILN.
     ENTER SFGETI USING "TAXES",   HOLD-TAXES.
     ENTER SFGETI USING "REPAIRS"  HOLD-REPAIRS.
     ENTER SFGETI USING "CLOSFEE"  HOLD-CLOSFEE.
     ENTER SFGETI USING "REALFEE"  HOLD-REALFEE.
```

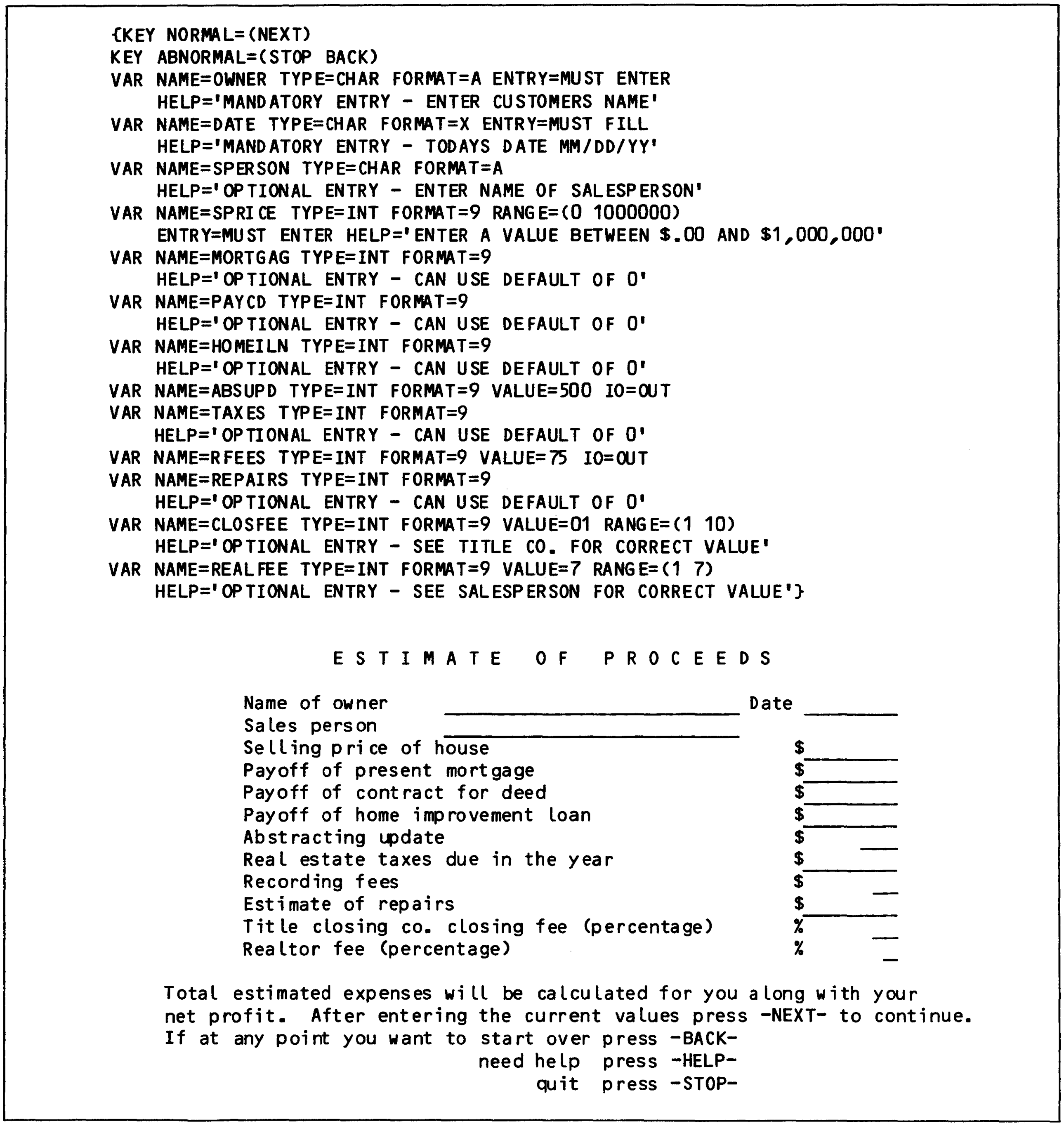
Figure D-7. COBOL Program ESTIMAT (Sheet 3 of 5)

```
 ACCEPTABLE-INPUT.
     COMPUTE ESTIMATED-EXPENSES =
             (HOLD-MORTGAG +
             HOLD-PAYCD    +
             HOLD-HOMEILN  +
             PANEL1-ABSUPD +
             HOLD-TAXES    +
             PANEL1-RFEES  +
             HOLD-REPAIRS  +
             (HOLD-CLOSFEE * .01 * HOLD-SPRICE) +
             (HOLD-REALFEE * .01 * HOLD-SPRICE)).
     COMPUTE ESTIMATED-CASH = HOLD-SPRICE - ESTIMATED-EXPENSES.
     MOVE ESTIMATED-CASH TO PANEL2-ECASH.
     MOVE ESTIMATED-EXPENSES TO PANEL2-EXPENSE.
     MOVE HOLD-SPRICE TO PANEL2-SPRICE.
     PERFORM RE-FILL-VARIABLES.
*
*     SFSWRI WRITES TO THE TERMINAL THE RESULTS FROM THE
*          CALCULATIONS IN ADDITION TO THE ORIGINAL DATA
*          RECEIVED FROM THE TERMINAL.
*
     ENTER SFSWRI USING "PANEL2", PANEL-VARIABLES.
*
*     SFSREA READS FROM THE TERMINAL A FUNCTION KEY.
*          IT DOES NOT RECEIVE ANY USER DATA BECAUSE "PANEL2"
*          DOES NOT CONTAIN ANY INPUT FIELDS.
*
     ENTER SFSREA USING "PANEL2", PANEL-VARIABLES.
*
*     SFGETK OBTAINS THE KEY.
*
     ENTER SFGETK USING KEY-TYPE, KEY-ORDINAL.
*
*     CHECK HERE FOR -NEXT- OR -BACK- KEYS.
*
     IF NEXT-KEY OR BACK-KEY
         GO TO START-OVER
     END-IF.
*
*     STATUS OF 1 SAYS GO TO LINE MODE.
*
     SET LINE-MODE TO TRUE.
     PERFORM CLOSE-PANELS.
     GO TO STOP-PROGRAM.
 START-OVER.
*
*     STATUS OF 0 SAYS KEEP IN SCREEN MODE.
*
     SET SCREEN-MODE TO TRUE.
     PERFORM CLOSE-PANELS.
     GO TO START-PROGRAM.
```

Figure D-7. COBOL Program ESTIMAT (Sheet 4 of 5)

```
 RE-FILL-VARIABLES.
     MOVE HOLD-SPRICE TO PANEL1-SPRICE.
     MOVE HOLD-MORTGAG TO PANEL1-MORTGAG.
     MOVE HOLD-PAYCD TO PANEL1-PAYCD.
     MOVE HOLD-HOMEILN TO PANEL1-HOMEILN.
     MOVE HOLD-TAXES TO PANEL1-TAXES.
     MOVE HOLD-REPAIRS TO PANEL1-REPAIRS.
     MOVE HOLD-CLOSFEE TO PANEL1-CLOSFEE.
 CLOSE-PANELS.
*
*     SFCLOS CLOSES THE PANELS.
*          PANEL-STATUS = 1 (TO SET TERMINAL TO LINE MODE)
*                       = 0 (TO SET TERMINAL TO SCREEN MODE)
*
     ENTER SFCLOS USING "PANEL2", PANEL-STATUS.
     ENTER SFCLOS USING "PANEL1", PANEL-STATUS.
 STOP-PROGRAM.
     STOP RUN.
```

Figure D-7. COBOL Program ESTIMAT (Sheet 5 of 5)



```
{KEY NORMAL=(NEXT)
KEY ABNORMAL=(STOP BACK)
VAR NAME=OWNER TYPE=CHAR FORMAT=A ENTRY=MUST ENTER
    HELP='MANDATORY ENTRY - ENTER CUSTOMERS NAME'
VAR NAME=DATE TYPE=CHAR FORMAT=X ENTRY=MUST FILL
    HELP='MANDATORY ENTRY - TODAYS DATE MM/DD/YY'
VAR NAME=SPERSON TYPE=CHAR FORMAT=A
    HELP='OPTIONAL ENTRY - ENTER NAME OF SALESPERSON'
VAR NAME=SPRICE TYPE=INT FORMAT=9 RANGE=(0 1000000)
    ENTRY=MUST ENTER HELP='ENTER A VALUE BETWEEN $.00 AND $1,000,000'
VAR NAME=MORTGAG TYPE=INT FORMAT=9
    HELP='OPTIONAL ENTRY - CAN USE DEFAULT OF 0'
VAR NAME=PAYCD TYPE=INT FORMAT=9
    HELP='OPTIONAL ENTRY - CAN USE DEFAULT OF 0'
VAR NAME=HOMEILN TYPE=INT FORMAT=9
    HELP='OPTIONAL ENTRY - CAN USE DEFAULT OF 0'
VAR NAME=ABSUPD TYPE=INT FORMAT=9 VALUE=500 IO=OUT
VAR NAME=TAXES TYPE=INT FORMAT=9
    HELP='OPTIONAL ENTRY - CAN USE DEFAULT OF 0'
VAR NAME=RFEES TYPE=INT FORMAT=9 VALUE=75 IO=OUT
VAR NAME=REPAIRS TYPE=INT FORMAT=9
    HELP='OPTIONAL ENTRY - CAN USE DEFAULT OF 0'
VAR NAME=CLOSFEE TYPE=INT FORMAT=9 VALUE=01 RANGE=(1 10)
    HELP='OPTIONAL ENTRY - SEE TITLE CO. FOR CORRECT VALUE'
VAR NAME=REALFEE TYPE=INT FORMAT=9 VALUE=7 RANGE=(1 7)
    HELP='OPTIONAL ENTRY - SEE SALESPERSON FOR CORRECT VALUE'}


                    E S T I M A T E   O F   P R O C E E D S

            Name of owner   ___________________________ Date ________
            Sales person    ___________________________
            Selling price of house                        $________
            Payoff of present mortgage                    $________
            Payoff of contract for deed                   $________
            Payoff of home improvement loan               $________
            Abstracting update                            $     ___
            Real estate taxes due in the year             $________
            Recording fees                                $      __
            Estimate of repairs                           $________
            Title closing co. closing fee (percentage)    %      __
            Realtor fee (percentage)                      %       _

     Total estimated expenses will be calculated for you along with your
     net profit.  After entering the current values press -NEXT- to continue.
     If at any point you want to start over press -BACK-
                                   need help  press -HELP-
                                        quit  press -STOP-
```

Figure D-8. PANEL1 Panel Definition File

```
{KEY ABNORMAL=(NEXT STOP BACK)
VAR NAME=OWNER IO=OUT
VAR NAME=DATE IO=OUT
VAR NAME=SPERSON IO=OUT
VAR NAME=SPRICE IO=OUT
VAR NAME=MORTGAG IO=OUT
VAR NAME=PAYCD IO=OUT
VAR NAME=HOMEILN IO=OUT
VAR NAME=ABSUPD IO=OUT
VAR NAME=TAXES IO=OUT
VAR NAME=RFEES IO=OUT
VAR NAME=REPAIRS IO=OUT
VAR NAME=CLOSFEE IO=OUT
VAR NAME=REALFEE IO=OUT
VAR NAME=EPRICE IO=OUT
VAR NAME=EXPENSE IO=OUT
VAR NAME=ECASH IO=OUT}

                E S T I M A T E   O F   P R O C E E D S

        Name of owner        __________________________ Date ________
        Sales person         __________________________
        Selling price of house                              $________
        Payoff of present mortgage                          $________
        Payoff of contract for deed                         $________
        Payoff of home improvement loan                     $________
        Abstracting update                                  $     ___
        Real estate taxes due in the year                   $________
        Recording fees                                      $      __
        Estimate of repairs                                 $________
        Title closing co. closing fee (percentage)          %      __
        Realtor fee (percentage)                            %       _

        Selling price of house                              $________
        Less total estimated expenses                       $________
        Total estimated cash to seller                      $________

                Press -NEXT- or -BACK- to continue
                                -STOP- to terminate
```

Figure D-9. PANEL2 Panel Definition File

PASCAL PROGRAM TRAIN

Figure D-10 shows a Pascal program called TRAIN. TRAIN will display a picture of a train on the screen.

Figure D-11 is the panel definition file for the TRAIN program.

```
PROGRAM EXAMPLE(OUTPUT);

CONST
  MAXSTR = 100;  (* MAXIMUM STRING SIZE *)

TYPE
  TERMMODE = (SCREEN,LINE,NOCLEAR);             (* TERMINATION STATUS *)
  IDENT    = PACKED ARRAY [1..7] OF CHAR;       (* IDENTIFIER *)
  STRING   = PACKED ARRAY[1..MAXSTR] OF CHAR;   (* DATA STRING *)

VAR
  BLANKS   : STRING;                            (* BLANK STRING *)
  INSTR    : STRING;                            (* INPUT STRING *)
  STATUS   : INTEGER;                           (* OPEN STATUS *)
  I        : INTEGER;                           (* LOOP INDEX *)

PROCEDURE SFCLOS(P:IDENT;MODE:TERMMODE); FORTRAN;
PROCEDURE SFOPEN(P:IDENT;VAR STATUS:INTEGER); FORTRAN;
PROCEDURE SFSSHO(P:IDENT;VAR OUTSTR:STRING;VAR INSTR:STRING); FORTRAN;

BEGIN  (* EXAMPLE *)
SFOPEN('TRAIN  ';,STATUS);
IF STATUS = 0 THEN
  BEGIN
  FOR I :=1 TO MAXSTR DO
    BLANKS[I] := ' ';

  SFSSHO('TRAIN  ',BLANKS,INSTR);
  SFCLOS('TRAIN  ',LINE);
  END
ELSE
  WRITELN('PANEL NOT FOUND.')
END.  (*EXAMPLE *)
```

Figure D-10. Pascal Program TRAIN

```
TRAIN
{    PANEL  NAME=TRAIN
     ATTR   DELIMITERS='//' PHYSICAL=ALTERNATE
     ATTR   DELIMITERS='!!' PHYSICAL=(ALTERNATE BLINK)
     BOX    TERMINATOR='*' WEIGHT=BOLD
     BOX    TERMINATOR='+' WEIGHT=MEDIUM
     VAR    COWCATCHER
}




                                  !@@@ @@ @@!
                                            !@@@!
                                                !@@!
                                                  !@!
                                           [&]    [ ]
                                          +---------+_
      *--*  *--*  *--*  *--*  *--*  *--*  |   SOO   |
      *--*~~*--*~~*--*~~*--*~~*--*~~*--*==+---------+)
       00    00    00    00    00    00    0000 0000 \\\
#############################################################################
```

Figure D-11. TRAIN Panel Definition File

STATIC LOADING OF PANELS E

By default, panels are dynamically loaded (by the Fast Dynamic Loader) when they are opened (by an SFOPEN object routine) and unloaded when they are closed (by an SFCLOS object routine). Some high-performance applications may wish to avoid the disk access requirements implied by dynamically loading panels. Also, some applications may wish to load more than the default maximum of 10 panels. This appendix describes how to load panels as part of the field length of the application program. If this is done, the panels may not be unloaded and will be memory resident for the duration of program execution.

Panel loading is controlled by a panel load table (PLT), which is a separate object module in the SFLIB system library. You can change the PLT by defining an alternate PLT (in Compass), assembling the alternate PLT, and copying the object module (LGO file) to a user program library. To use the alternate PLT, insert the following command into the load sequence:

```
    LDSET,LIB=SFLIB/USERLIB.
```

If the redefined PLT is in USERLIB, it will be used instead of the default PLT in SFLIB.

PANEL LOAD TABLE FORMAT

The PLT has a 2-word header. The low-order 12 bits of the first word contain the number of table entries which follow the header. The low-order 12 bits of the second word contain the number of panels currently in memory.

Following the header are one or more 2-word entries. The number of entries determines how many panels can be in memory at once.

The high-order 42 bits of the first word contain the panel name in display code (seven characters). The high-order bit (bit 59) of the second word is set if the panel is statically loaded. The low-order 18 bits contain the address of the panel in memory.

The following procedure compiles a program along with a PLT which statically loads the panel MYPANEL. The user-supplied PLT allows up to two other panels to be dynamically loaded.

```
.PROC,MYPLT.
REWIND,*.
PDU,MYPANEL.
FTN5,I=MYPROG,L=0.
COMPASS,I=PLT,L=0.
LDSET,LIB=SFLIB/PANELIB.
LOAD,LGO.
NOGO,MINE.
RETURN,MYPANEL,MYPROG,PLT.
REVERT,NOLIST.
.DATA,MYPANEL


                         TEST PANEL


                     ENTER ANYTHING: #_
.DATA,MYPROG
      PROGRAM MYPROG
      CHARACTER*1 S
      CALL SFOPEN('MYPANEL',I)
      IF (I.EQ.0) THEN
        CALL SFSREA('MYPANEL',S)
        CALL SFCLOS('MYPANEL',1)
      ENDIF
      END
.DATA,PLT
          IDENT   PLT
          ENTRY   PLT
*
*         THIS CODE WILL FORCE THE CYBER LOADER TO STATICALLY
*         LOAD *MYPANEL*.  SPACE IS LEFT TO DYNAMICALLY LOAD
*         UP TO TWO OTHER PANELS.
*
PLT       VFD     60/3
          VFD     60/3
          VFD     60/7LMYPANEL
          VFD     1/1,41/0,18/=XMYPANEL
          VFD     60/0
          VFD     60/0
          VFD     60/0
          VFD     60/0
          END
```

MIGRATION GUIDELINES F

Panels and application programs intended to be migrated to future systems should use the following guidelines to minimize the conversion effort.

PANEL SYNTAX

The PANEL program currently accepts certain syntactical variants which do not conform to the documentation. For example, semicolons may be omitted between successive statements on the same line. Since these variations may be corrected at any time, we recommend that you follow the documented syntax rules.

The {} characters which begin and end the panel declaration section should be written as the only characters on their respective lines.

PANEL FORMAT

References to function keys should be confined to a known part of each panel image (such as the bottom), and KEY statements should be placed in a known part of the panel declarations. The reason for this is that future products may allow the use of more terminal independent selection devices which may include function keys as a subset. The application developer may wish to modify the panels to take advantage of this higher level service.

STANDARD LANGUAGES

Application programs should be written in ANSI standard FORTRAN, COBOL, and Pascal languages. Consult each language reference manual for a list of potential problem areas.

CHARACTER SETS

For maximum portability, application programs should use only the default character set for the programming language; in other words, the 6-bit display code set.

If 6-bit display code is not suitable, the next most portable character set is the NOS 7-bit ASCII set, which uses exactly two display code character positions for each single ASCII character. References to variables or items containing such data should compare or move them as a whole rather than character by character. All such data declarations or references to individual characters will have to be converted manually.

OPTIMIZATIONS

Static loading of panels by redefining the default panel load table should not be used by programs intended for migration.

TERMINAL KEY LABELS G

NOS 2 now supports screen formatting on almost any display terminal. By using the terminal definition utility (TDU), the user can define terminal attributes for use with full-screen products. Seven terminals are system-defined for full-screen use. They are:

- CDC Viking 721
- CDC 722
- Tektronix 4115†
- Zenith Z19/Heathkit H19†
- DEC VT100†
- Lear Siegler ADM3A†
- Lear Siegler ADM5†

The logical and physical attributes you can define are dependent on your terminal´s capabilities. This information must be obtained from the terminal hardware reference manual, as explained in section 5.

Table G-1 lists the Viking 721 application and CDC standard function keys. In this manual, we use these Viking 721 physical key labels when referring to the logical function performed.

Across from each Viking 721 key is the key or sequence of keys that must be used on the other system-defined terminals to generate the same function. If more than one key must be used, they are shown with a plus between them indicating they should be entered consecutively. These Viking 721 application keys and CDC standard function keys perform functions defined by the application. Using TDU, you can change the attributes of any of these function keys.

Table G-2 lists the physical display attributes that can be defined by TDU and which attributes are available on the seven system-defined terminals. As with the logical functions, these physical capabilities vary with different terminals. If an attribute is defined but not available, it may be mapped into another attribute, which is listed in the table, or it may be ignored ("No").

Some terminals require the user to press NEXT or its equivalent (NEWLINE or RETURN, for example) after each function key. You can, however, press a function key or function key sequence several times before you press NEXT. You cannot press several different function keys or sequences before you press NEXT. If you press a different function key or sequence, it is ignored.

†When using this terminal, change the network control character (ct) to something other than ESC. This terminal uses escape sequences for function key definitions. To change the network control character, enter: TRMDEF,CT=value (for more information, refer to the NOS Reference Set, Volume 3).

Table G-1. Function Keys on System-Defined Terminals (Sheet 1 of 3)

CDC Viking 721	CDC 722	Lear Siegler ADM3A	Lear Siegler ADM5	Tektronics T4115	Zenith Z19	Digital VT100
F1	F1 + NEWLINE	ESC + 1 + RETURN	ESC + 1 + RETURN	F1	F1 + RETURN	KEYPAD 1 + RETURN
F2	F2 + NEWLINE	ESC + 2 + RETURN	ESC + 2 + RETURN	F2	F2 + RETURN	KEYPAD 2 + RETURN
F3	F3 + NEWLINE	ESC + 3 + RETURN	ESC + 3 + RETURN	F3	F3 + RETURN	KEYPAD 3 + RETURN
F4	F4 + NEWLINE	ESC + 4 + RETURN	ESC + 4 + RETURN	F4	F4 + RETURN	KEYPAD 4 + RETURN
F5	F5 + NEWLINE	ESC + 5 + RETURN	ESC + 5 + RETURN	F5	F5 + RETURN	KEYPAD 5 + RETURN
F6	F6 + NEWLINE	ESC + 6 + RETURN	ESC + 6 + RETURN	F6	F6(BLUE) + RETURN	KEYPAD 6 + RETURN
F7	F7 + NEWLINE	ESC + 7 + RETURN	ESC + 7 + RETURN	F7	F7(RED) + RETURN	KEYPAD 7 + RETURN
F8	F8 + NEWLINE	ESC + 8 + RETURN	ESC + 8 + RETURN	F8	F8(WHITE) + RETURN	KEYPAD 8 + RETURN
F9	F9 + NEWLINE	ESC + 9 + RETURN	ESC + 9 + RETURN	CTRL A		KEYPAD 9 + RETURN
F10	F10 + NEWLINE	ESC + 0 + RETURN	ESC + 0 + RETURN	CTRL S		
F11	F11 + NEWLINE	ESC + : + RETURN	ESC + : + RETURN	CTRL D		
F12		ESC + - + RETURN	ESC + - + RETURN	CTRL F		
F13		ESC + [+ RETURN	ESC + [+ RETURN			
F14		ESC +] + RETURN	ESC +] + RETURN			
F15		ESC + ^ + RETURN				
F16		ESC + | + RETURN				

Table G-1. Function Keys on System-Defined Terminals (Sheet 2 of 3)

CDC Viking 721	CDC 722	Lear Siegler ADM3A	Lear Siegler ADM5	Tektronics T4115	Zenith Z19	Digital VT100
SHIFT F1	SHIFT F1 + NEWLINE	ESC + SHIFT 1 + RETURN	ESC + SHIFT 1 + RETURN	SHIFT F1	SHIFT F1 + RETURN	PF1 + RETURN
SHIFT F2	SHIFT F2 + NEWLINE	ESC + SHIFT 2 + RETURN	ESC + SHIFT 2 + RETURN	SHIFT F2	SHIFT F2 + RETURN	PF2 + RETURN
SHIFT F3	SHIFT F3 + NEWLINE	ESC + SHIFT 3 + RETURN	ESC + SHIFT 3 + RETURN	SHIFT F3	SHIFT F3 + RETURN	PF3 + RETURN
SHIFT F4	SHIFT F4 + NEWLINE	ESC + SHIFT 4 + RETURN	ESC + SHIFT 4 + RETURN	SHIFT F4	SHIFT F4 + RETURN	PF4 + RETURN
SHIFT F5	SHIFT F5 + NEWLINE	ESC + SHIFT 5 + RETURN	ESC + SHIFT 5 + RETURN	SHIFT F5	SHIFT F5 + RETURN	KEYPAD − + RETURN
SHIFT F6	SHIFT F6 + NEWLINE	ESC + SHIFT 6 + RETURN	ESC + SHIFT 6 + RETURN	SHIFT F6	SHIFT F6 + RETURN	KEYPAD , + RETURN
SHIFT F7	SHIFT F7 + NEWLINE	ESC + SHIFT 7 + RETURN	ESC + SHIFT 7 + RETURN	SHIFT F7	SHIFT F7 + RETURN	KEYPAD ENTER + RETURN
SHIFT F8	SHIFT F8 + NEWLINE	ESC + SHIFT 8 + RETURN	ESC + SHIFT 8 + RETURN	SHIFT F8	SHIFT F8 + RETURN	KEYPAD . + RETURN
SHIFT F9	SHIFT F9 + NEWLINE	ESC + SHIFT 9 + RETURN	ESC + SHIFT 9 + RETURN	CTRL Q		
SHIFT F10	SHIFT F10 + NEWLINE	ESC + SHIFT 0 + RETURN	ESC + SHIFT 0 + RETURN	CTRL W		
SHIFT F11	SHIFT F11 + NEWLINE	ESC + SHIFT : + RETURN	ESC + SHIFT : + RETURN	CTRL E		
SHIFT F12		ESC + SHIFT − + RETURN	ESC + SHIFT − + RETURN	CTRL R		
SHIFT F13		ESC + SHIFT [+ RETURN	ESC + SHIFT [+ RETURN			
SHIFT F14		ESC + SHIFT] + RETURN	ESC + SHIFT] + RETURN			
SHIFT F15		ESC + SHIFT ^ + RETURN				
SHIFT F16						

Table G-1. Function Keys on System-Defined Terminals (Sheet 3 of 3)

CDC Viking 721	CDC 722	Lear Siegler ADM3A	Lear Siegler ADM5	Tektronics T4115	Zenith Z19	Digital VT100
NEXT	NEWLINE or CR	RETURN	RETURN	RETURN	RETURN	RETURN
HELP		ESC + h + RETURN	ESC + h + RETURN			
BACK		ESC + k + RETURN	ESC + k + RETURN			
STOP or CTRL T	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN
FWD		ESC + f + RETURN	ESC + f + RETURN			
BKW		ESC + b + RETURN	ESC + b + RETURN			
UP		ESC + u + RETURN	ESC + u + RETURN			
DOWN		ESC + d + RETURN	ESC + d + RETURN			
SHIFT HELP		ESC + H + RETURN	ESC + H RETURN			
SHIFT BACK		ESC + K + RETURN	ESC + K + RETURN			
SHIFT STOP	CTRL T + NEWLINE	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN	CTRL T + RETURN
SHIFT FWD		ESC + F + RETURN	ESC + F + RETURN			
SHIFT BKW		ESC + B + RETURN	ESC + B + RETURN			
SHIFT UP		ESC + U + RETURN	ESC + U + RETURN			
SHIFT DOWN		ESC + D + RETURN	ESC + D + RETURN			
SHIFT CLEAR	CTRL X					

Table G-2. Attributes Available on Supported Terminals

Attribute	CDC 721	CDC 722	Lear Siegler ADM3A	Lear Siegler ADM5	Tektronix 4115	Zenith/ Heathkit	DEC VT100
ALTERNATE	Yes	No	No	No	No	No	Yes
BLINK	Yes	No	No	No	Yes	No	Yes
INVERSE	Yes	No	No	No	Yes	Yes	Yes
UNDERLINE	Yes	No	No	No	Yes	No	Yes
BLACK	No	No	No	No	Yes	No	No
RED	No	No	No	No	No	No	No
GREEN	Yes	No	No	No	No	No	No
BLUE	No	No	No	No	No	No	No
YELLOW	No	No	No	No	No	No	No
MAGENTA	No	No	No	No	No	No	No
CYAN	No	No	No	No	No	No	No
WHITE	No	Yes	Yes	Yes	No	Yes	Yes
line drawing	Yes	No	No	No	No	Yes	Yes
WEIGHT	Yes	No	No	No	No	Yes	Yes

INDEX

COMMENT SHEET

MANUAL TITLE: CDC NOS Version 2 Screen Formatting Reference Manual

PUBLICATION NO.: 60460430 REVISION: B

NAME:____________________________________

COMPANY:____________________________________

STREET ADDRESS:____________________________________

CITY:__________________ STATE:________ ZIP CODE:________

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

□ Please Reply □ No Reply Necessary

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

CUT ALONG LINE

LD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

LD FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION