NOS Version 2 Screen Formatting

Reference Manual

CONTROL DATA

NOS Version 2
Screen Formatting

Reference Manual

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60460430

Manual History

Revision	System Version/PSR Level	Date
A	Manual released. This revision reflects NOS 2.2 at PSR level 596.	October, 1983
B	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, 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.	October, 1984
C	This manual reflects NOS 2.4.2 at PSR level 642. This revision includes the Queued Terminal Record Manager (QTRM) screen formatting feature and support of the IBM 3270 terminal.	September, 1985
D	This manual reflects NOS 2.6.1 at PSR level 700. This revision includes the SFATTR object routine, which allows the application program to change the attributes of a panel variable at run time. Additional technical and editorial corrections have been made.	April, 1988

Contents

Figures

Tables

About This Manual

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 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 Version 2 Reference Set, Volumes 2 and 3.

Organization

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

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 Viking 721 NEXT key, although the key has different names on different terminals (such as RETURN, NEWLINE, and SEND).

The notation CTRL/x directs you to press the control key (which is labeled CTRL, CNTL, CNTRL, or similar characters) on the terminal and, while holding this key down, press the key specified by x. For example, CTRL/H means press and hold the control key while you press the H key.

Submitting Comments

There is a comment sheet at the back of this manual. You can use it to give us your opinion of the manual's usability, to suggest specific improvements, and to report errors. If the comment sheet has already been used, mail your comments to:

Control Data
Technology and Publications Division ARH219
4201 North Lexington Avenue
St. Paul, MN 55126-6198

Please indicate whether you would like a response.

If you have access to SOLVER, an online facility for reporting problems, you can use it to submit comments about the manual. When entering your comments, use NS2 as the product identifier. Include the name and publication number of the manual.

If you have questions about the packaging and/or distribution of a printed manual, write to:

Control Data
Literature Distribution Services
308 North Dale Street
St. Paul, MN 55103

Or call (612) 292-2101. If you are a Control Data employee, calll (612) 292-2100.

CYBER Software Support Hotline

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help not provided in the documentation, or find the product does not perform as described, call us at one of the following numbers. A support analyst will work with you.

From the USA and Canada: (800) 345-9903
From other countries: (612) 851-4131

Related Manuals

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

Control Data Publications	Publication Number
COBOL Version 5 Reference Manual	60497100
FORTRAN Version 5 Reference Manual	60481300
Network Access Method Version 1 Host Application Programming Reference Manual	60499500
NOS Full Screen Editor	60460420
NOS Version 2 Reference Set, Volume 2, Guide to System Usage	60459670
NOS Version 2 Reference Set, Volume 3, System Commands	60459680
Pascal Version 1.1 Reference Manual	60497700

You may also want to consult the NOS System Information Manual. It is an online manual that briefly describes all NOS and NOS product manuals. You access this manual by logging into NOS and entering the command EXPLAIN.

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 Version 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.

Introduction 1

Introduction 1

The NOS screen formatting feature provides full-screen input and output capabilities for NOS procedures and for FORTRAN 5, COBOL 5, 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 most character mode display terminals, although some terminals have capabilities that make screen formatting more usable. For more information about these needed capabilities, refer to chapter 5. Terminal capabilities and keys can be defined for full-screen use by using the terminal definition utility (TDU). A number of terminals are already system-defined for use of screen formatting and Full Screen Editor (FSE). These terminals are listed in chapter 5.

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 panel 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?

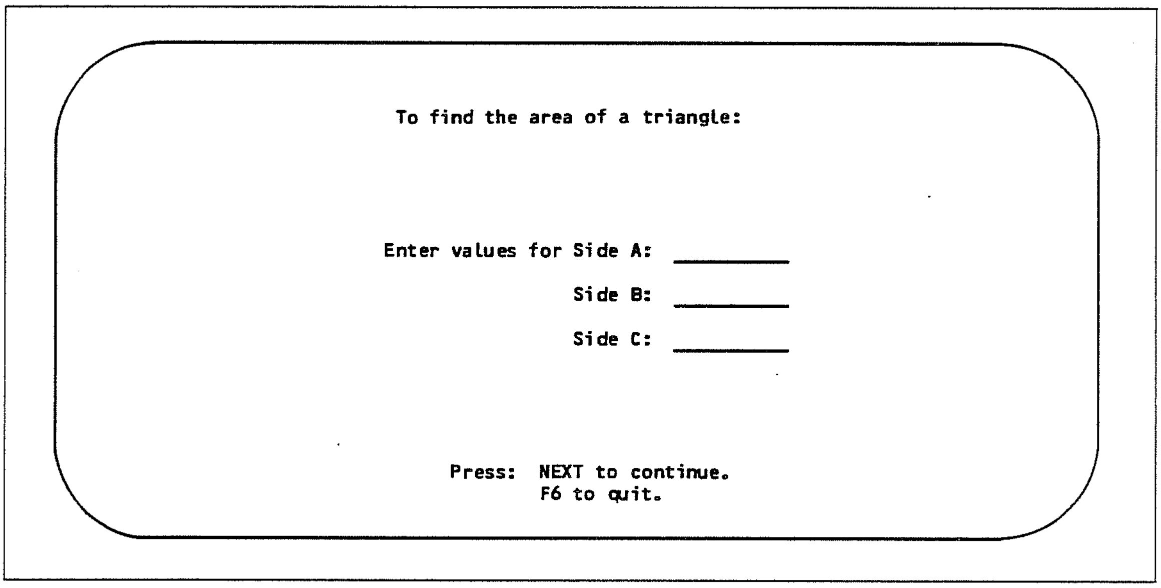
Interactive job processing can be divided into two types: line mode and screen mode. As the name implies, line mode processing handles terminal input and output one line at a time. In response to a system or program prompt, you type in one line of data and submit the line for processing by pressing the 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 does not display the next prompt until the current line is properly entered. 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 you are satisfied that all entries are correct, press the NEXT key (or whatever key is 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 as a normal or abnormal exit, or as a match advance key.



```
                To find the area of a triangle:



               Enter values for Side A: ________
                                   Side B: ________
                                   Side C: ________



                    Press:  NEXT to continue.
                            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

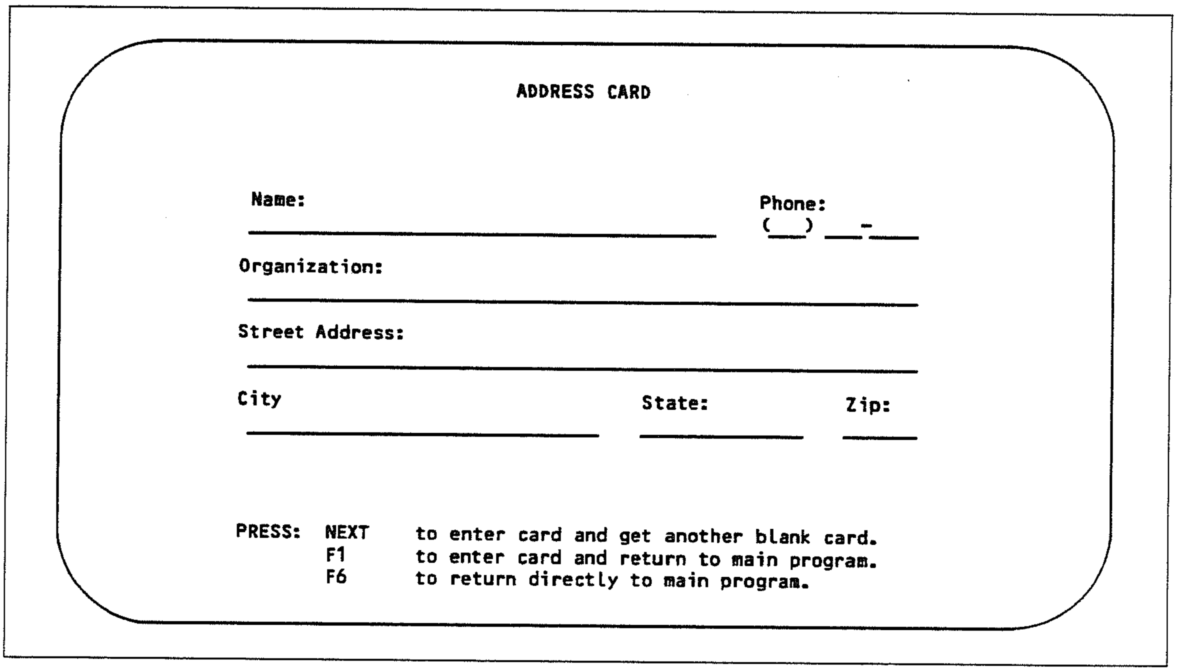
For NOS procedure writers, screen mode parameter display is an automatic system feature. Once you have entered a SCREEN command (as described in the NOS Version 2 Reference Set, Volume 3), the system automatically presents all subsequent parameter displays in a system-defined, full-screen format. Full-screen 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 chapter 4.

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.)

Panel creation is the function of the panel definition utility (PDU), described in chapter 2. That chapter describes the declaration statements and formats used to create a panel definition within an ordinary NOS text file. It also describes 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 chapter 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, or converting the character strings entered into a different data type.



```
                        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

Screen Formatting for Queued Terminal Record Manager (QTRM) Application Programs

Directly connected multi-user network application programs, written in FORTRAN 5 or COBOL 5, and using the QTRM interface to the Application Interface Program (AIP), have a screen formatting capability similar to standard screen formatting as described in the first four chapters of this manual.

The creation and maintenance of panels is identical to that described for standard screen formatting.

To accommodate both the multi-user environment and the necessary structuring of network input/output, a different screen formatting library is provided for QTRM screen formatting application programs.

QTRM screen formatting usage is described in chapter 6.

Panel Definition Utility 2

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 chapter 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.

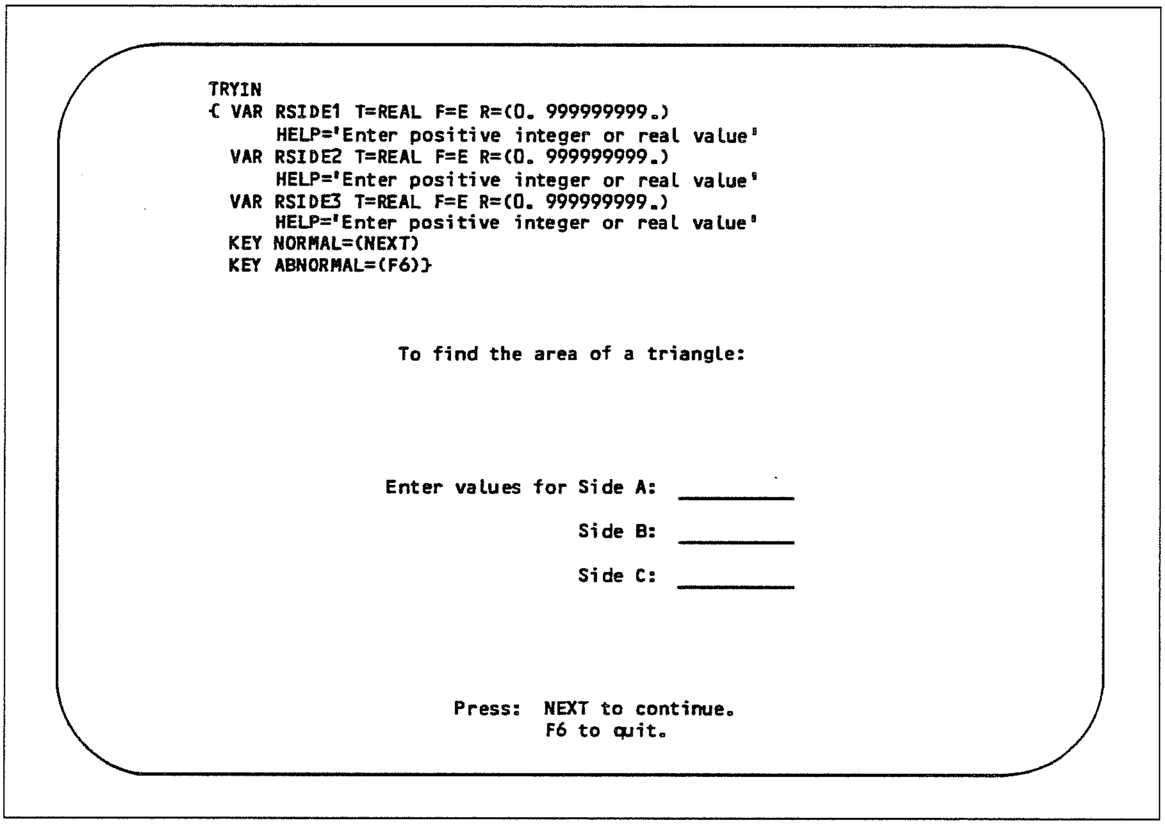
Figure 2-1 shows an example of a panel definition file. The title line contains the file name, TRYIN. The declaration section follows the title line and consists of a block of definition statements enclosed in braces. The image section consists of all file lines following the last line of the declaration 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 Version 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 continue.
                                    F6 to quit.
```

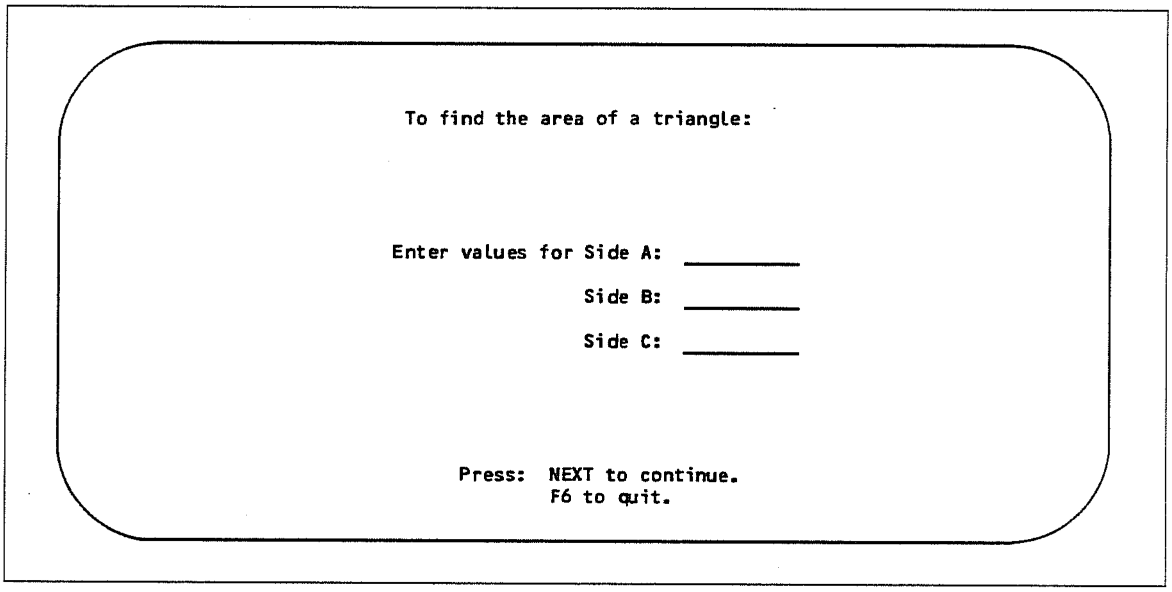
Figure 2-1. Panel Definition File

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 continue.
                             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 ("). For example, H= 'Please enter your selection'. When the HELP key is selected for the appropriate keyword, the message enclosed in apostrophes is written to the screen. Comments are inserted into the declaration section by enclosing them in quotation marks (" "). PDU ignores all data enclosed in quotation marks.

The following is a sample declaration section that defines five variable fields:

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

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.

Physical and Logical Attributes

The variable field statement allows you to specify physical or logical display attributes that will be associated with the variable field. 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. For this reason, use logical types to describe input/output fields. 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 has 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 briefly describes 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. The maximum number of variable fields is 255.

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 chapter.

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' or D	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$_n$) or P	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 attr$_n$ is specified, the attribute list must be enclosed in parentheses. The list can contain one or more of the following physical attributes:

attr$_n$	Description
ALTERNATE or ALT	Alternate intensity character display.
BLINK	Blinking character display.
INVERSE or INV	Inverse video display.
UNDERLINE or UND	Underlined character string.

Parameter	Description
LOGICAL=attribute or L	Specifies a logical display attribute to be associated with the delimiter. The LOGICAL parameter cannot be specified if the PHYSICAL parameter is specified. attribute 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 INVERSE)
```

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 solid or broken, 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 looks 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.
- When used in overlay panels (see the PANEL statement), lines containing only box characters are not cleared when the overlay is written.

The format of the BOX statement is:

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

The BOX statement parameters are:

Parameter	Description
TERMINATOR='*' or T	Defines the line termination character. * 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 or W	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$_n$) or P	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 or L	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 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 application 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 chapter 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. Therefore, if you define any keys, make sure at least one is designated as a normal or abnormal exit key (preferably one of each).

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 message

  ```
  Please enter
  ```

  if the field has had no data input to it or the message

  ```
  Please correct
  ```

  if the user has entered an invalid response.

NOTE

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=(key1 key2 ... keyn)
    ABNORMAL=(key1 key2 ... keyn)
    MATCH=(key1 key2 ... keyn)
    HELP=(key1 key2 ... keyn)
```

The KEY statement parameters are:

Parameter	Description
NORMAL=(key$_1$ key$_2$ ... key$_n$) or N	Specifies the function key or keys that cause a normal return to the application program. A normal return means that data the user entered is checked against the validation requirements specified in the associated VAR statements. 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 (the NEXT key cannot be shifted). Key names that can be specified include any of the programmable function keys (F1 through F24) and any of the following CDC standard function keys: BACK BKW DATA DOWN EDIT FWD HELP NEXT STOP UP Refer to appendix G for more information on these keys.
ABNORMAL=(key$_1$ key$_2$ ... key$_n$) or A	Specifies the function key or keys that cause an abnormal return to the application program. An abnormal return causes input to be returned to the application program immediately with no validation checking. Key names that can be specified are the same as for the NORMAL parameter.
MATCH=(key$_1$ key$_2$ ... key$_n$) or M	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$_n$) or H	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 entire screen.

When an overlay panel is displayed, lines in the overlay panel containing variable fields or constant data overwrite the corresponding lines in the current screen display. Blank lines (and those containing only boxes) 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 or N	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 or T	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 or N	Specifies the name of the table. The name can be from one to seven alphanumeric characters.
ROWS=number or R	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.

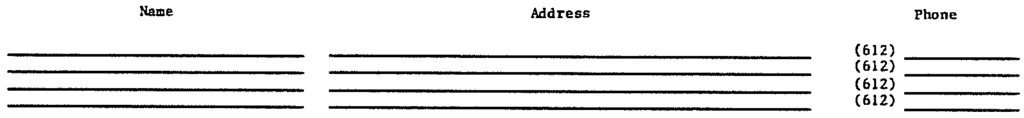
The following example shows a simple table definition as it might appear in the declaration section of a panel definition file:

```
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:

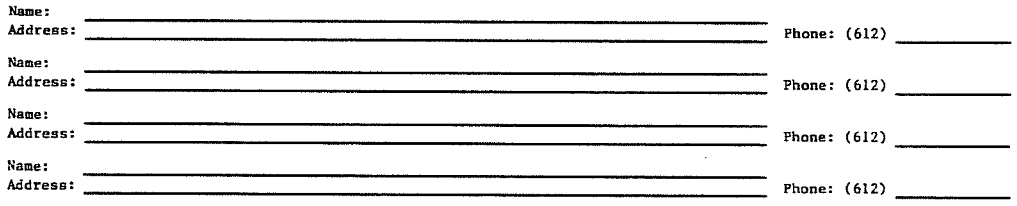
	NAME	ADDR	PHONE
Row 1	name,1	addr,1	phone,1
Row 2	name,2	addr,2	phone,2
Row 3	name,3	addr,3	phone,3
Row 4	name,4	addr,4	phone,4

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:



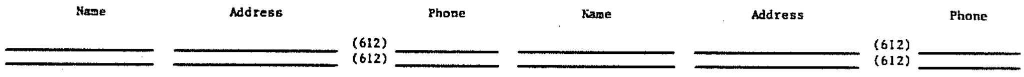
```
         Name                          Address                   Phone

_____________________  __________________________________  (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=(string1 string2 ... stringn) ...
    RANGE=(low high) ...
    LOGICAL=attr
    PHYSICAL=(attr1 attr2 ... attrn) ...
    ENTRY=condition ...
    IO=status ...
    HELP=string
```

The VAR statement parameters are:

Parameter	Description
NAME=fieldname or N	Specifies a variable field name, one to seven characters long.
TYPE=type or T	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 or V	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=code or F	Specifies the acceptable input format for the variable. This parameter does not reformat or otherwise affect the contents of the field. code 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. (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.)
YMD or Y	Allow date entry in YY/MM/DD format.
MDY or M	Allow date entry in MM/DD/YY format.
DMY or D	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.

Parameter — **Description**

FORMAT=code or F (Continued)

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

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

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) or R

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).

LOGICAL=attr or L

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.

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

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.

Parameter	Description
ENTRY=(condition) or E	Specifies special conditions pertaining to entry of the variable field. These conditions will be checked before any further specified validation (such as a MATCH list or RANGE validation). Values that can be specified for the ENTRY condition are any one of, or combination of, the following:

condition	Description
MUST ENTER	The user must enter something (even blanks will suffice) into the field every time the panel is read by calling SFSREA or as a result of calling SFSSHO.
MUST FILL	If the user enters anything into the field (other than blanks) the field must be full of non-blank characters with no leading, embedded, or trailing blanks.
MUST CONTAIN	The field must contain some non-blank character, either entered by the user or from the OUTSTRING supplied by the application itself. In combination with MUST ENTER this means that the field must have some non-blank character in it and that the user must have typed something into the field. Used with MUST FILL will require that the field be filled and cannot be filled with blanks. A combination of MUST ENTER, MUST FILL, and MUST CONTAIN means that the user must enter something, it can not be blanks, and it must fill the field.
UNKNOWN	The user may enter an asterisk when unsure of what to enter. Note that this will skip all validation for the field, which includes disregarding MUST FILL and MUST CONTAIN as well as any further validation such as MATCH or RANGE validation.

Parameter **Description**

`IO=status or IO`

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 or H`

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.

NOTE

A maximum of 255 variable fields may be defined in a panel.

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 (the first line on the screen), and the screen cursor moves to the beginning of the variable field in error. 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 prompts appear in the message field;

```
Please correct
```

or, if there has been no data input:

```
Please enter
```

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.

Do not use the first line of a panel for input fields, since diagnostic messages generated by the screen formatting subroutines are displayed left-justified on the first line. You may use the first line of the panel for output variable or constant fields. If input information is displayed on the first line, any diagnostic messages returned will overwrite this information.

As noted in the variable section, a maximum of 255 variables are allowed per panel. In addition, a panel may have up to 256 constant fields. A constant field is a string of characters with no more than one consecutive embedded blank character. For example, "A Δ CONSTANT" is only one field, "A ΔΔ CONSTANT" is two fields. (Δ denotes one blank space.)

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. Variable fields on the same line should be delimited by spaces. You may position the fields anywhere in the panel except the first line, which is used for messages. Variable fields must appear in the image section in the same order as defined in the declaration section. 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.

NOTE

The first line of the image section cannot contain any input variable fields, as that line is used for help messages by screen formatting.

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 underscore 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 five-character variable field that looks like this in the definition file:

  ```
  _ _ _ _ _
  ```

  looks like this when displayed on a terminal with a video underline capability:

  ```
  _________
  ```

When using terminals that do not support the underline attribute, you can identify the input fields by using delimiting characters that 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 appear in the variable fields and are 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 be able to reproduce only the definition characters you used to define lines in the panel image. In the latter case, the image and the resulting panel 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 that 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 and the error message PANEL–CAN'T OPEN FILE 0 is issued.
`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. It 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
A	Add a record to a user library.
C	Create a new user library.
D	Delete a record from 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.
R	Replace a record in a user library.

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

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 FORTRAN 5, COBOL 5, 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 the screen formatting object routines interact with NOS. This section 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 FORTRAN 5, COBOL 5, 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 Version 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 that 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. Model names for the system-defined terminals are described in 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 is entered, the screen formatting object routines, when called in an executing program, set the terminal to screen mode and 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 FORTRAN 5, COBOL 5, 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(p1,p2,p3)
```

objrtn The six-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 p1 p2 p3.
```

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 converts them to integer or real variables, as required.

Object Routines

This section 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
SFATTR	Allows an application program to change the attributes associated with a panel variable field.
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.
SFGETK	Determines the last function key pressed.
SFGETN	Returns the current terminal model name specified on the SCREEN command.
SFGETP	Determines the cursor position when a function key was pressed.
SFGETR	Returns the real value of a single variable field.
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.

SFATTR (fieldname,atrord,ordstat)

The STATTR objective routine allows a screen formatting application program to change the attributes associated with a panel variable field at run time.

The SFATTR parameters are:

Parameter	Type	Description
fieldname	char	The name of the variable field as defined to the PDU.
atrord	int	The requested new field type ordinal for the desired attributes. Possible values for atrord are:

Atrord	Protected Field	Guarded Field	Logical Field Type	(Suggested physical mapping for CDC 721 to be in the TDUFILE)
0	Y	-	INPUT	UNDERLINE
1	N	Y		
2	N	N		
3	Y	-	OUTPUT	NONE
4	N	Y		
5	N	N		
6	Y	-	ITALICS	BLINK
7	N	Y		
8	N	N		
9	Y	-	TITLE	ALTERNATE
10	N	Y		
11	N	N		
12	Y	-	MESSAGE	USER[1]
13	N	Y		
14	N	N		
15	Y	-	ERROR	INVERSE
16	N	Y		
17	N	N		
18	Y	-	INPUT 2	USER[2]
19	N	Y		
20	N	N		

[1] The MESSAGE is currently used by screen formatting on any error messages and has no physical attributes defined in the released TDU definitions.

[2] In keeping with the practice of input fields having an underline attribute, it is suggested that some underline with blink or some other combination that includes underline be used here.

Parameter	Type	Description
Atrord (continued)		

Atrord	Protected Field	Guarded Field	Logical Field Type	(Suggested physical mapping for CDC 721 to be in the TDUFILE)
21	Y	-	OUTPUT2	USER
22	N	Y		
23	N	N		
24	Y	-	ITALICS2	USER
25	N	Y		
26	N	N		
27	Y	-	TITLE2	USER
28	N	Y		
29	N	N		
30	Y	-	MESSAGE2	USER
31	N	Y		
32	N	N		
33	Y	-	ERROR2	USER
34	N	Y		
35	N	N		

Parameter	Type	Description
ordstat	int	The variable to which SFATTR will return a value indicating the results of an attempt to change the attributes associated with a variable field. Possible values for ordstat are the current field types listed above for 0 - 35 or:

ordstat	Description
-3	If the requested new ordinal was incorrect.
-2	If the field was not found in the panel.
-1	If the new attribute requested was not in the panel.
0	If unprotected with no logical type, for example, INPUT.
3	If protected with no logical type, for example, OUTPUT.

Examples:

```
CALL SFATTR('FIELD1',NEWORD,IORDST)

ENTER SFATTR USING"FIELD1"NEW-ORD ORD-STAT.

SFATTR('FIELD1Δ',NEWORD,ORDSTAT)
```

NOTE

Δ denotes a space. Pascal requires a reserved seven-character variable.

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 ('ASCII8Δ');
```

NOTE

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

Δ denotes a space. Pascal requires a reserved seven-character variable.

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 returns 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: YMD 83/1/31 MDY 1/31/83 DMY 31/1/83
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 ('FIELD1Δ',I);
```

NOTE

Δ denotes a space. Pascal requires a reserved seven-character literal.

SFGETK (type,value)

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

The SFGETK parameters are:

Parameter	Type	Description
type	int	The variable to which SFGETK returns an integer indicating whether the last function key pressed is a CDC standard function key or a programmable function key. The options for type are:

type	Description
0	Programmable function key.
1	CDC standard function key.

Parameter	Type	Description
value	int	The variable to which SFGETK returns an integer indicating the last function key pressed. For programmable function keys, the value corresponds to the key ordinals (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:

value	Key
1	NEXT
2	BACK
3	HELP
4	STOP
5	DOWN
6	UP
7	FWD
8	BKW
9	EDIT
10	DATA

Examples:

```
CALL SFGETK (TRMTYPE, PROGKEY)
```

```
ENTER SFGETK USING TRMTYPE. PROGKEY.
```

```
SFGETK (TRMTYPE, PROGKEY)
```

SFGETN (modelname)

The SFGETN object routine returns the current model name as specified on the NOS SCREEN or LINE command (as defined with the TDU model name parameter in the TDU definition being used for the given terminal).

The SFGETN parameters are :

Parameter	Type	Description
modelname	char	One- to six-character model name as specified on the NOS SCREEN or LINE command. If no terminal has been successfully defined to NOS via the SCREEN or LINE command, then a string of blanks is returned to the calling program.

Examples :

```
CALL SFGETN(MODEL)

ENTER SFGETN USING MODEL

SFGETN(MODEL);
```

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 returns a value indicating the field name of the variable field in which the cursor was last positioned.
index	int	The variable to which SFGETP returns 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 returns 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);
```

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 returns 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 ('FIELD2Δ',R);
```

NOTE

Δ denotes a space. Pascal requires a reserved seven character literal.

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 returns a value indicating the results of the attempt to open a panel. A value other than 0 indicates the panel could not be opened. Possible values for status are:

Status	Significance
0	The panel was successfully opened.
1	The panel was not found.
2	The panel capsule was incorrectly formatted, probably due to panel definition errors.
3	Too many panels are 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 was 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 one- to seven-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 ('PLAINVΔ',1,2);
```

NOTE

Δ denotes a space. Pascal requires a reserved seven-character literal.

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 was not 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 writes are cleared from the screen by SFSREA.

The SFSREA parameters are:

Parameter	Type	Description
panelname	char	The name of the panel used for input.
instring	char	The variable to which SFSREA returns 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 chapter.)

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 that are 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 writes. 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 chapter.)
instring	char	The variable to which SFSSHO returns 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 chapter.)

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 writes 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 chapter.)

Examples:

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

ENTER SFSWRI USING "MYPANEL" OUT-STRING.

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

NOS Procedures in Screen Mode 4

NOS Procedures in Screen Mode 4

NOS screen formatting allows you 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 that can increase the usability of your procedures. Becoming familiar with the screen mode display features will help you write procedures that most effectively use 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 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 chapter describes how procedures are executed in screen mode and tells you how to write procedures for screen mode display.

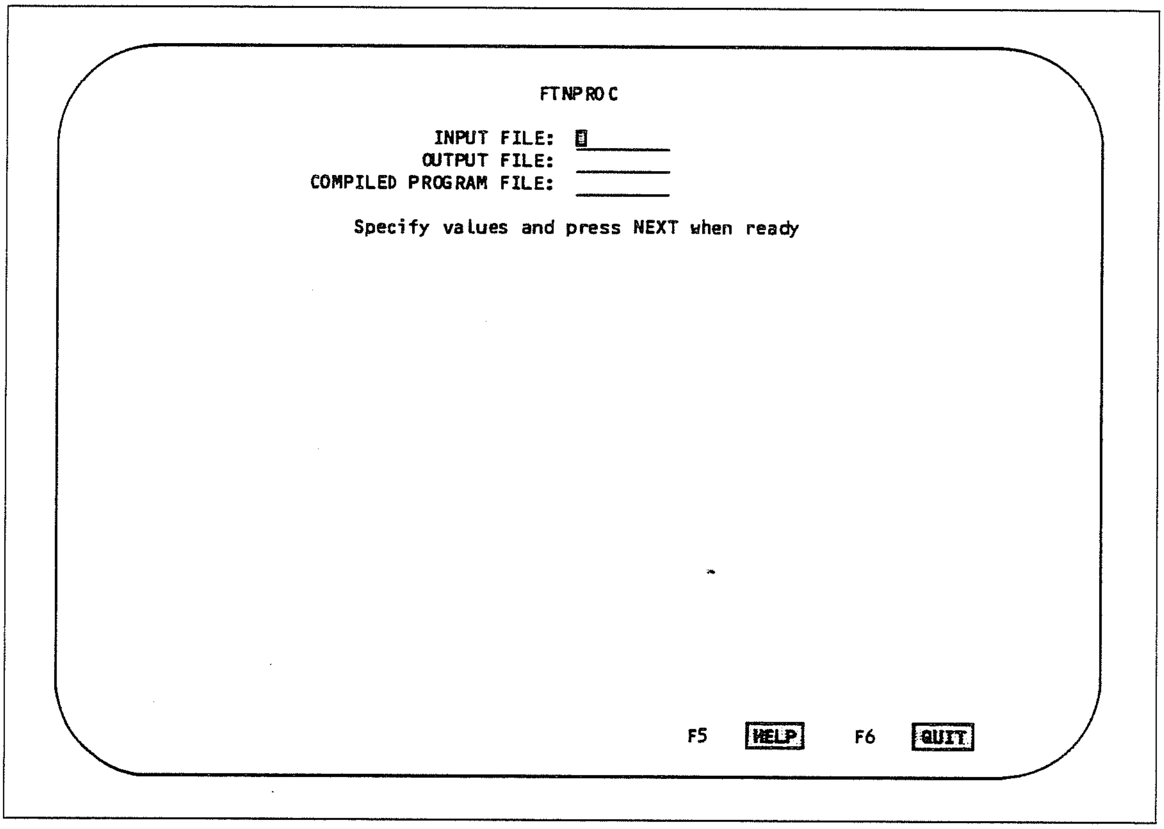
Procedure Execution

Screen mode display of NOS procedure parameters requires no special call format. When you request prompting for interactive procedure parameters, the parameters are displayed either in line mode or in screen mode, depending on the terminal status. If you 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 you call 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.

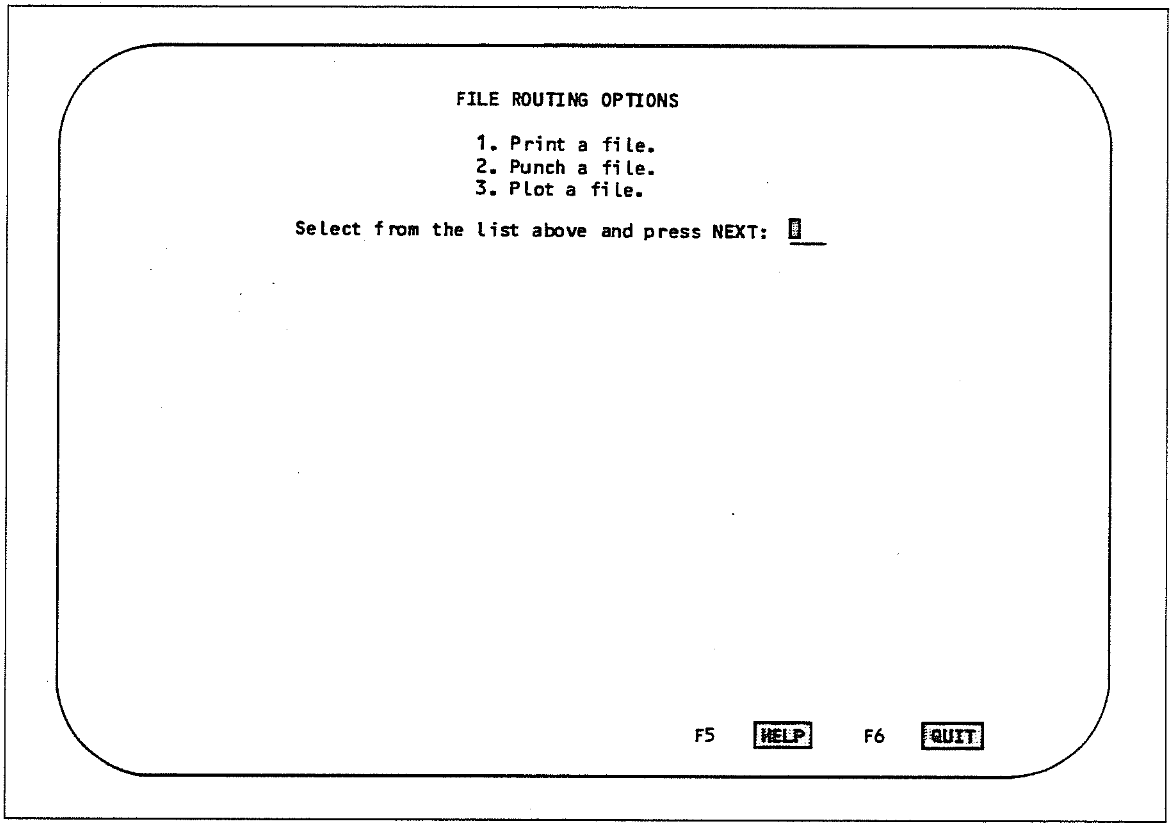
The parameter displays for a single procedure occupy up to nine screens of display text. You can page forward and backward through the screen displays by pressing designated function keys. While paging through the parameter displays, you can enter or modify parameter values in any order. To move from one parameter field to the next, press 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, move 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

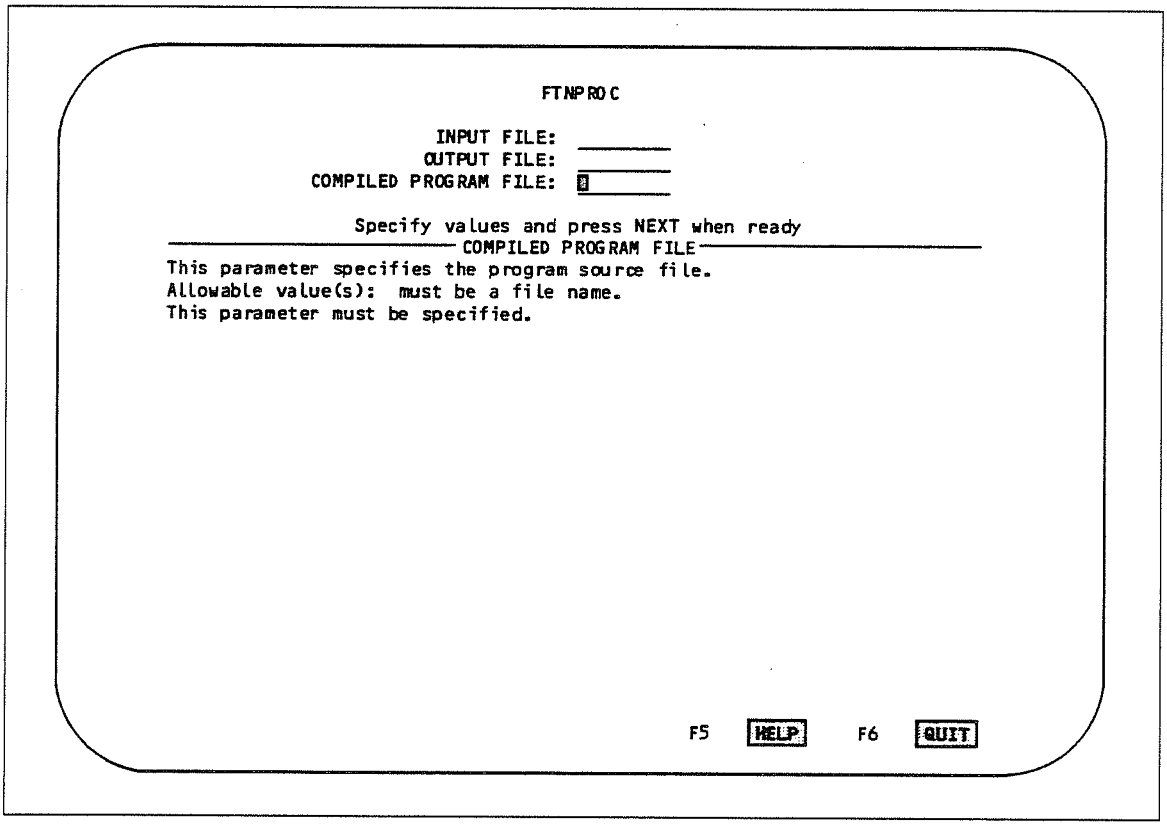


```
                         FILE ROUTING OPTIONS

                           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, you 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 you 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 you enter all required parameters, execute 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 you omit a required parameter or enter an incorrect value, the system prompts for a correct value before initiating execution of the procedure.

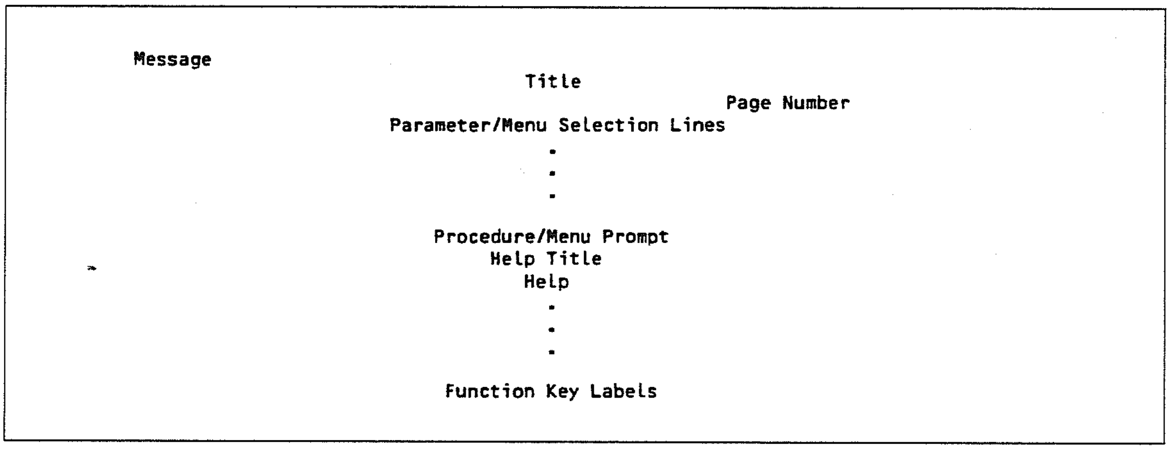


```
                              FTNPROC

                       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. For a complete discussion of directives and options available for interactive procedures, refer to chapter 4 of the NOS Version 2 Reference Set, Volume 3, System Commands.

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 one to 40 characters.

The following message is displayed when an invalid value is 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 one 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 one to 40 characters.

Parameter/Menu Selection Lines

The page number line is followed by a variable number of lines that 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 ranges 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 second 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:
Parameter[Description]=	Description:

Regardless of which format is used, each parameter prompt is followed by a one- 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
*Fm . . n *Pm . . n	A value equal to the maximum length as specified by n. n may be up to 7.
*Am . . n	A value equal to the maximum length as specified by n. n may be up to 40.
*K	A value equal to the length of the parameter name.
*Sm . . n	A value equal to the maximum length of the set as specified by n. n may be up to 40.
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)	Input file: _ _ _ _ _ _ _
I'File to copy'=(*F)	File to copy: _ _ _ _ _ _ _
R'Rewind (Y or N)'=(Y, N)	Rewind (Y or N): _
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 one to 40 characters.

Help Title

The help title line appears on the screen only when help text is 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 (if possible on the terminal being used), 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.
F7	EXECUTE	Displayed and active only if a .F7 directive is included in the procedure. Causes the current procedure to begin execution. The NEXT or RETURN key, ordinarily used for procedure execution, now advances the cursor to the next field if .F7 is displayed.

You can replace the default function key labels or activate F7 key (procedure execution) by using the .Fx directive. The .Fx directive format is:

```
  .Fx,string
```

x
Specifies an integer value from 1 to 7, corresponding to a function key from F1 to F7.

string
Specifies a character string, from one to six characters. If omitted, no label appears in the inverse box, as shown for the key in question, F1-F6. However, for F7, the default label is EXECUTE if string is omitted.

With the exception of .F7, 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 one- to 40-character text string that appears on the screen. message can consist of both uppercase and lowercase characters.

Terminal Definition Utility 5

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.

The NOS system has terminal definitions for many terminals. These definitions are records on the direct access file TDUFILE under user name LIBRARY. You may access this file and copy any of the records containing terminal definitions. You may then use this copy to modify the definition to meet your particular needs.

The first record on the file TDUFILE is TDUIN. This record is a template of a terminal definition file with the statement values left blank. If you plan to use the information in this chapter to create your own terminal definition file, you may want to use a copy of TDUIN. Embedded in the statements are explanations to help you fill out the file with the values for your terminal. Refer to Terminal Definition File later in this section for more information on TDUIN.

In the following list, the name of records containing the terminal definitions on TDUFILE are listed on the left. The model name used in the SCREEN or LINE command is shown in the middle column. The names of the corresponding terminals are listed across from each record name.

Record Name	Model Name	Terminal Name
TDUIN		Template file to be used to create your own file.
TDU721	721	Viking 721
TDU722	722	CDC 722
TDU7223	72230	CDC 722-30
TDU3270	3270	IBM 3270
TDUPCCN	PCCONN	IBM PC/CONNECT
TDUVT10	VT100	DEC VT100
TDUT415	T4115	TEKTRONIX T4115
TDUZ19	Z19	ZENITH Z19/Z29 or Heathkit H19
TDUADM3	ADM3A	LEAR SIEGLER ADM3A
TDUADM5	ADM5	LEAR SIEGLER ADM5
TDUVKX3	721V3	Viking 721 with Version 3.0 firmware.
TDU721T	721T	Viking 721 with type ahead, touch panel, and automatic tabbing (for screen formatting)
TDU722T	722T	CDC 722 with type ahead
TDUVT1T	VT100T	DEC VT100 with type ahead
TDUT41T	T4115T	TEKTRONIX T4115 with type ahead
TDUZ19T	Z19T	ZENITH Z19/Z29 with type ahead
TDUAD3T	ADM3AT	LEAR SIEGLER ADM3A with type ahead
TDUAD5T	ADM5T	LEAR SIEGLER ADM5 with type ahead
TDUVK3T	721V3T	Viking 721 with Version 3.0 firmware and type ahead
TDU723T	72230T	CDC 722-30 using type ahead
TDUPC11	PCON11	IBM PC/CONNECT 1.1
TDUPC12	PCON12	IBM PC/CONNECT 1.2
TDUPC13	PCON13	IBM PC/CONNECT 1.3
TDUMACC	MACCON	Macintosh/CONNECT
TDUMC11	MCON11	Macintosh/CONNECT 1.1
TDU924	TV924	Televideo 924
TDU950	TV950	Televideo 950
TDU955	TV955	Televideo 955
TDU924T	TV924T	Televideo 924 using type ahead
TDU950T	TV950T	Televideo 950 using type ahead
TDU955T	TV955T	Televideo 955 using type ahead
TDUSUN	SUN160	SUN 160 Workstation
TDUSUNT	SUN16T	SUN 160 Workstation using type ahead
TDU910I	CDC910	CDC 910 Workstation (IRIS)
TDU910T	CD910T	CDC 910 Workstation (IRIS) using type ahead
TDU910C	CD910C	CDC 910 Workstation (CLOVER)
TDU91CT	CD91CT	CDC 910 Workstation (CLOVER) using type ahead

Terminal Capabilities

Any display terminal with certain minimal capabilities, which can be defined using the TDU utility, will work with any full-screen product. Refer to your terminal hardware reference manual to verify that your terminal has the required 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 that 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 also have the following attributes:

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

The following terminal attributes are also desirable:

- Eight to 32 function keys.
- Function keys that transmit a unique, identifying character sequence followed by (or including) 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 included in the 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 commands:

```
ATTACH,TDUFILE/UN=LIBRARY

COPYBR,TDUFILE,TDUIN
```

This copies the first record of TDUFILE (TDUIN) to a new file name TDUIN. 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 your copy of TDUIN. 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.

Your 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. For more information, refer to the NOS Full Screen Editor manual.

Compile your terminal definition file using the TDU utility and store it on TERMLIB. This load capsule is 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, check whether someone 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 in the TERMLIB file, 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       keyword1=value1 keyword2=value2...
                     keywordn=valuen
```

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 ellipsis (...) 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 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.
32(8)	The ESC character as an octal number.
ESC	The 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 character strings ('721','a'), a character (ESC), and a hexadecimal number (12(16)).

If you are going to use 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 record on file TDUFILE used 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

HAS_PROTECT    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 TYPE parameters describe the cursor positioning movement. For example:

```
MOVE_PAST_SIDE       TYPE = WRAP_ADJACENT_NEXT
```

The selectable values for TYPE are predefined for you in the TDUIN file.

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. May not exceed 64.

COLUMNS

The number of columns on the terminal. May not exceed 25.

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**

Screen/line mode transition

Describes terminal attributes set and cleared when a full screen application is entered or exited.

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.

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. For example:

```
HELP   LABEL = 'HELP' IN = (RS 5C (16) )
```

LABEL is optional.

Output

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

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

BELL_NAK        OUT=(BEL)
```

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.

An IN/OUT statement may be split into two statements; an IN statement and an OUT statement. This is needed if your terminal sends a different sequence to the host to perform a certain function than is sent by the host to the terminal when that function is performed. For example, the IN/OUT statement

```
TAB_FORWARD     IN = (    )

                OUT = (    )
```

may need to be split as follows:

```
TAB_FORWARD               IN = (    )

TAB_FORWARD               OUT = (    )
```

The statements may be split, if desired, even when the values are not different. Do not, however, combine any other IN and OUT statements.

Required Capabilities

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

CLEAR_PAGE_STAY or CLEAR_PAGE_HOME	
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), and a stop-function key (such as CTRL/T). All statements that are required will be identified as such in their descriptions in the TDUIN file.

Terminal Attribute Statements

The following statements may be used to describe terminal characteristics:

<table>
<tr><th>Statement</th><th>Parameter</th><th>Description</th></tr>
<tr><td>MODEL_NAME</td><td></td><td>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.</td></tr>
<tr><td></td><td>VALUE=name</td><td>The model name may be a one to six alphanumeric character string. Lowercase letters are translated to uppercase.</td></tr>
<tr><td>COMMUNICATIONS</td><td></td><td>Identifies the type of communication the terminal uses. Required statement.</td></tr>
<tr><td></td><td>TYPE=type</td><td>type refers to the terminal protocol.
<table>
<tr><th>type</th><th>Protocol</th></tr>
<tr><td>ASYNCH</td><td>Asynchronous</td></tr>
<tr><td>SYNCH</td><td>Synchronous</td></tr>
</table></td></tr>
<tr><td>CURSOR_POS_ENCODING</td><td></td><td>Tells how the cursor position output sequence is encoded. Most terminals fall in one of the categories below. Required statement.</td></tr>
<tr><td></td><td>TYPE=encoding</td><td>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:
<p>axbyc</p>
<p>All terminals will have an a, x, and y at least. The value of encoding is interpreted as follows:</p>
<table>
<tr><th>encoding</th><th>Description</th></tr>
<tr><td>BINARY_CURSOR</td><td>The cursor positioning sequence is of the format:
<pre>
a (x+bias) (y+bias)   or
a (y+bias) (x+bias)
</pre></td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Statement</th><th>Parameter</th><th>Description</th></tr>
<tr><td>CURSOR_POS_ENCODING
(Continued)</td><td></td><td>Required statement.
<table>
<tr><th>encoding</th><th>Description</th></tr>
<tr><td>ANSI_
CURSOR</td><td>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</td></tr>
<tr><td>CDC721_
CURSOR</td><td>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)</td></tr>
<tr><td>IBM3270_
CURSOR</td><td>The cursor positioning sequence is of the format ba where ba is the 3270 buffer address.</td></tr>
</table></td></tr>
<tr><td></td><td>BIAS=number</td><td>Specifies an integer to be added to the x and y values. The usual number is 32, which is the value of the 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.</td></tr>
</table>

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
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. VALUE is 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 statements have either VALUE=TRUE or VALUE=FALSE parameters. These are required parameters.

Statement	Description
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.
CLEARS_WHEN_CHANGE_SIZE	Changing the screen size causes the screen to be cleared. FALSE if your terminal supports only one screen size.
FUNCTION_KEY_LEAVES_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.
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.
HAS_PROTECT	The PROTECT_BEGIN and PROTECT_END sequences can be used to define protected areas on the screen.

Statement	Description
HOME_AT_TOP	The CURSOR_HOME sequence sends the cursor to the top left of the screen rather than to the bottom.
MULTIPLE_SIZES	There is more than one SET_SIZE statement.
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_TAB_STOPS	The terminal supports tabbing to settable or predefined tab stops (like typewriter tabs).
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.
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 allows you to make changes you cannot see on the screen unless you clear the page.

Cursor Positioning Statements

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

Parameter	Description
HOME_NEXT	The cursor moves to the home position.
SCROLL_NEXT	The terminal scrolls all characters on the screen (up, down, or sideways).
STOP_NEXT	The cursor refuses to move beyond the edge.
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
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. If the TYPE value for this statement is SCROLL_NEXT, then the last character on the bottom right corner of the panel is not sent to the terminal screen even though it is on the text file.
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.
MOVE_PAST_BOTTOM	Describes what happens when the cursor is moved past the bottom of the screen using the cursor movement keys.
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.

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
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. This may be empty if only one size is available.
ROWS=number	The number (an integer) of rows (lines) to which the terminal will be set.

Initialization Output Statements

Statement	Description
LINE_INIT	This sequence is sent whenever the LINE command is executed.
SCREEN_INIT	This sequence is sent whenever the SCREEN command is executed.

Screen/Line Mode Transition Statements

Statement	Description
SET_LINE_MODE	This sequence is sent whenever the terminal switches from screen mode to line mode. This should reverse the SET_SCREEN_MODE configuration.
SET_SCREEN_MODE	This sequence is sent whenever the terminal switches from line mode to screen mode. This is where the configuration is set for running screen formatting applications.

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
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.)
CURSOR_DOWN	Moves the cursor down one line. Required statement.
CURSOR_HOME	Moves the cursor to the home position. No full-screen application will function acceptably without this. This is a required statement.
CURSOR_LEFT	Moves the cursor left one position. Required statement.
CURSOR_RIGHT	Moves the cursor right one position. Required statement.

Statement	Description
CURSOR_UP	Moves the cursor up one line. Required statement.
DELETE_CHAR	Deletes a single character at the current position, shifting the present text to the left.
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.
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.
ERASE_CHAR	Erases the character at the current position, moving the cursor left one position.
ERASE_END_OF_FIELD	Erases from the current position to the end of the unprotected field. Leaves the cursor where it is.
ERASE_END_OF_LINE	Erases from the current position to the end of the line. Leaves the cursor where it is. Full-screen products function better with this capability.
ERASE_END_OF_PAGE	Erases the screen from the current cursor position to the bottom of the screen.
ERASE_FIELD_BOF	Erases the current unprotected field. Moves the cursor to the start of that unprotected field.
ERASE_FIELD_STAY	Erases the current unprotected field. Leaves the cursor where it is.
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_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_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.

Statement	Description
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.
ERASE_UNPROTECTED	Erases all the unprotected character positions on the screen.
INSERT_CHAR	Inserts a single blank character at the current position, shifting present text to the right.
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.
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_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.
INSERT_MODE_TOGGLE	Switches between insert and overstrike mode.
RESET	Resets the terminal hardware. The terminal must be reinitialized.
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_FORWARD	Tabs to the next tab stop or unprotected field.
TAB_SET	Sets a tab stop at the current position.

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	Shifted CDC Standard Function Keys
BACK	BACK_S
BKW	BKW_S
DATA	DATA_S
DOWN	DOWN_S
EDIT	EDIT_S
FWD	FWD_S
HELP	HELP_S
NEXT	STOP_S
STOP	UP_S
UP	

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, if possible, should be defined for all of the keys 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	Shifted Programmable Function Keys
F1	F1_S
f2	f2_s
f3	f3_s
f4	f4_s
f5	f5_s
f6	f6_s
f7	f7_s
f8	f8_s
f9	f9_s
f10	f10_s
f11	f11_s
f12	f12_s
f13	f13_s
f14	f14_s
f15	f15_s
F16	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
BELL_ACK	Ring the alternate bell.
BELL_NAK	Ring the bell on an error. Default is ASCII BEL (7).
DISPLAY_BEGIN	Enable the display so characters received show on the screen.
DISPLAY_END	Disable the display.
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.
PRINT_BEGIN	Enable the printer so characters received print.
PRINT_END	Disable the printer.
PROTECT_ALL	Protect every character position on the screen.
RETURN	Move the cursor to the beginning of the current line.

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

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

```
FINE LINE DRAWING SEQUENCES:
LD_FINE_BEGIN
LD_FINE_END
HORIZONTAL AND VERTICAL CHARACTERS:
LD_FINE_HORIZONTAL
LD_FINE_VERTICAL
BOX CORNER CHARACTERS:
LD_FINE_UPPER_LEFT
LD_FINE_UPPER_RIGHT
LD_FINE_LOWER_LEFT
LD_FINE_LOWER_RIGHT
INTERSECTION CHARACTERS:
LD_FINE_UP_T
LD_FINE_DOWN_T
LD_FINE_LEFT_T
LD_FINE_RIGHT_T
LD_FINE_CROSS
MEDIUM LINE DRAWING SEQUENCES:
LD_MEDIUM_BEGIN
LD_MEDIUM_END
HORIZONTAL AND VERTICAL CHARACTERS:
LD_MEDIUM_HORIZONTAL
LD_MEDIUM_VERTICAL
BOX CORNER CHARACTERS:
LD_MEDIUM_UPPER_LEFT
LD_MEDIUM_UPPER_RIGHT
LD_MEDIUM_LOWER_LEFT
LD_MEDIUM_LOWER_RIGHT
INTERSECTION CHARACTERS:
LD_MEDIUM_UP_T
LD_MEDIUM_DOWN_T
```

```
LD_MEDIUM_LEFT_T
LD_MEDIUM_RIGHT_T
LD_MEDIUM_CROSS
BOLD LINE DRAWING SEQUENCES:
LD_BOLD_BEGIN
LD_BOLD_END
HORIZONTAL AND VERTICAL CHARACTERS:
LD_BOLD_HORIZONTAL
LD_BOLD_VERTICAL
BOX CORNER CHARACTERS:
LD_BOLD_UPPER_LEFT
LD_BOLD_UPPER_RIGHT
LD_BOLD_LOWER_LEFT
LD_BOLD_LOWER_RIGHT
INTERSECTION CHARACTERS:
LD_BOLD_UP_T
LD_BOLD_DOWN_T
LD_BOLD_LEFT_T
LD_BOLD_RIGHT_T
LD_BOLD_CROSS
```

Default Key Definitions for the Full Screen Editor

You may use the statement described in this section for defining the default function key sequences used by FSE. These keys may also be defined within your FSEPROC. For more information on FSEPROC, refer to the FSE User's Guide. They must be defined in one of these two places.

This statement can only be 250 characters long, including all parameters and their values. Use the ellipsis (. . .) for continuation within the statement. If 250 characters is insufficient when defining all the function keys and labels desired, additional statements with the same parameter name may be used.

Statement	Description
APPLICATION_STRING	Sets the default function key sequences used by the full screen editor.

Parameter	Description
NAME	The value is FSEKEYS, which is recognized by FSE as defining the function key commands.
OUT	The value will be a series of SET KEY FSE commands that FSE should perform when the associated function key is pressed. These SET KEY commands are separated by semi-colons. You may want to use previously defined variable strings, but remember that the 250 maximum length includes the entire sequence length.

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 ZZZZTRM 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 CDC VIKING 721 TERMINAL                          "

"   VARIABLES                                                                     "
    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)

"   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/')
    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/')

"   MODEL NAME AND COMMUNICATION TYPE                                             "
    model_name          value = '721'
    communications      type  = asynch

"   END OF INFORMATION SPECIFICATION                                              "
    end_of_information  in    = (0)
```

```
"   CURSOR POSITIONING INFORMATION                                           "
    cursor_pos_encoding      bias  = (32)    type = cdc721_cursor
    cursor_pos_column_first  value = TRUE
    cursor_pos_column_length value = (0)
    cursor_pos_row_length    value = (0)
    cursor_pos_begin         in    = (1e(16) 4d(16) 1f(16))
    cursor_pos_begin         out   = (stx)
    cursor_pos_second        out   = (7E(16) soh)

"   CURSOR MOVEMENT INFORMATION                                              "
    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          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          type  = wrap_adjacent_next
    char_past_left           type  = wrap_adjacent_next
    char_past_last_position  type  = wrap_adjacent_next

"   TERMINAL ATTRIBUTES                                                      "
    clears_when_change_size  value = TRUE
    function_key_leaves_mark value = FALSE
    has_hidden               value = TRUE
    has_protect              value = TRUE
    home_at_top              value = TRUE
    multiple_sizes           value = TRUE
    tabs_to_home             value = FALSE
    tabs_to_tab_stops        value = TRUE
    tabs_to_unprotected      value = TRUE
```

```
"   SCREEN SIZES                                                              "
    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                                           "
    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       out = (scroll_mode  enable_auto_cr  clear_all_tabs  ...
         pop_fn_keys)

"   TERMINAL CAPABILITIES                                                     "
    delete_char         inout = (rs 4e(16))
    delete_line_stay    inout = (rs 51(16))
    erase_char          inout = (1f(16))
    erase_end_of_line   inout = (vt)
    erase_field_stay    inout = (rs 59(16))
    erase_line_bol      inout = (rs 5D(16))
    erase_page_home     inout = (ff)
    insert_char         inout = (rs 4f(16))
    insert_line_stay    inout = (rs 52(16))
    tab_backward        inout = (rs 0b(16))
    tab_clear           inout = (rs dc2 'X')
    tab_clear_all       inout = (clear_all_tabs)
    tab_forward         inout = (ht)
    tab_set             inout = (rs dc2 'W')

"   MISCELLANEOUS TERMINAL SEQUENCES                                          "
    bell_nak            out = (bel)
    output_begin        out = (disable_protect)
    output_end          out = (enable_protect)
    protect_all         out = (rs 'G')
```

```
"   PROGRAMMABLE FUNCTION KEY INPUT INFORMATION                                  "
    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                                  "
    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                                                      "
    alt_begin           out = (fs)
    alt_end             out = (gs)
    blink_begin         out = (so etx)
    blink_end           out = (si)
    hidden_begin        out = (rs dc2 '[')
    hidden_end          out = (rs dc2 5C(16))
    inverse_begin       out = (start_inverse)
    inverse_end         out = (stop_inverse)
    protect_begin       out = (rs dc2 'I')
    protect_end         out = (rs dc2 'J')
    underline_begin     out = (start_underline)
    underline_end       out = (stop_underline)

"   LOGICAL ATTRIBUTE SPECIFICATIONS                                               "
    error_begin         out = (start_inverse)
    error_end           out = (stop_inverse)
    input_text_begin    out = (start_underline)
    input_text_end      out = (stop_underline)
    italic_begin        out = ()
    italic_end          out = ()
    message_begin       out = ()
    message_end         out = ()
    output_text_begin   out = ()
    output_text_end     out = ()
    title_begin         out = ()
    title_end           out = ()
```

```
"   LINE DRAWING CHARACTER SPECIFICATION                                    "
    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)
    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)
    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                      "
    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)

"   END OF TERMINAL DEFINITION FILE FOR CDC VIKING 721 TERMINAL
```

Queued Terminal Record Manager Screen Formatting 6

Queued Terminal Record Manager Screen Formatting 6

The NOS queued terminal record manager (QTRM) screen formatting provides multi-user full screen input and output capabilities for directly connected network applications which use the QTRM interface.

The QTRM screen formatting object routine library (QSFLIB) allows a FORTRAN 5 or COBOL 5 QTRM application program to provide a formatted screen capability. The QTRM screen formatting interface is similar to the standard NOS screen formatting object routine library described in the first chapters of this manual. The names and parameters of common subroutine calls are unchanged. However, there are differences in how QTRM handles some of the common routines. There are also some new routines for QTRM. Panels are still defined under IAF, and panels used in standard NOS screen formatting are not affected by QTRM screen formatting.

This chapter on QTRM screen formatting is intended for a FORTRAN 5 or COBOL 5 network application's programmer. It is assumed you are already familiar with QTRM and NOS screen formatting. For specific information on these products, see the Network Access Method Version 1 Host Application Programming Reference Manual and the beginning chapters of this manual.

Using QTRM Screen Formatting

Using the panel definition utility (PDU), you define and store panels in libraries so that they can be used by application programs. The QTRM screen formatting object routines described in this section allow FORTRAN 5 or COBOL 5 programs to retrieve panels from libraries and use them to perform terminal input/output operations through the QTRM interface.

None of the QTRM screen formatting object routines are directly involved in the entry of input or the display of output at the terminal, but rather they queue this data for subsequent use by the QTRM I/O operations. Some of the routines manage the input/output queues. Other routines deal with related tasks such as data conversion and determining cursor position.

System Considerations

The system library QSFLIB provides the object routines for QTRM screen formatting. This library must be linked to the FORTRAN 5 or COBOL 5 QTRM application program using the CYBER Loader.

The following NOS commands compile, load, and execute a FORTRAN 5 program using QTRM and the QTRM screen formatting routines. In the example that follows, the source program is called TETSOR, and the absolute binary program created is called TESTABS.

```
REWIND,*.
FTN5,I=TESTOR,L=LISTING.
LOAD,LGO.
LDSET,LIB=QSFLIB/NETIO.
NOGO,TESTABS.
TESTABS.
```

Screen and Line Modes

The QTRM application program must determine from the user what terminal is being used. The terminal model specified must correspond to a terminal definition that is either system-resident or that appears as a TDU compiled terminal definition on file TERMLIB on user name=LIBRARY.

To specify the mode and model to the QTRM screen formatting routines, the application must call the SFMODE routine with the designated mode and terminal model name.

An application attempting to open a panel will receive an error code if the terminal specified by the user has not been successfully declared to the QTRM screen formatting routines via a call to SFMODE.

Transparent Input and Output

Multi-message transparent mode must be enabled prior to receiving input from a connection so that the input received is in a format suitable for processing by the QTRM screen formatting routines. When transparent input processing is completed, multi-message transparent mode must be disabled.

This is achieved by calling the routine QTTIP. For more information on QTTIP and multi-message transparent mode, refer to the Network Access Method Version 1 Host Application Programming Reference Manual.

Character Sets

When you start the application, it sends a call to QTOPEN, the character set value must be equal to 11. This specifies that 8-bit codes are packed five bytes per 60-bit word, that output data is transmitted in transparent mode, and that input data may be in either normal or transparent mode. You may switch character sets in the application at any time after netting on by calls to QTSUP, but you must have the character set equal to 11 when doing any processing associated with the QTRM screen formatting routines.

Queueing of Input/Output Data

The QTRM screen formatting routines do not directly send output or receive input from a given terminal connection. Instead, two queues are maintained internally within the QTRM screen formatting routines: a GET queue and a PUT queue, which are used to temporarily store input and output data, respectively.

The GET and PUT queues are managed by the two routines SFDQUE and SFNQUE, which enable the application program to store and retrieve information in these queues by dynamically allocating the necessary storage via the common memory manager (CMM).

Object Routines

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

Object routine SFSSHO, which is found in the standard NOS screen formatting object routine library (SFLIB), is not available to the QTRM application, as it is not practical for a multi-user application to issue output and wait for input from a single connection in this environment.

Table 6-1 lists the QTRM screen formatting object routines with a brief description of each. Detailed descriptions follow the table for routines that are additional or act differently when called by a QTRM application from the standard screen formatting routines by the same name. These routines are indicated in the table by a footnote.

Table 6-1. QTRM Screen Formatting Object Routines

Object Routine	Description
SFCLOS[1]	Unloads a panel after use by the application program.
SFCSET	Specifies the code set that the application program uses for input and output data.
SFDQUE[1]	Extracts information from specified queue into the specified buffer.
SFGETI	Returns the integer value of a single variable field.
SFGETK	Determines the last function key pressed.
SFGETN	Returns the current terminal model name.
SFGETP	Determines the last cursor position when a function key was pressed.
SFGETR	Returns the real value of a single variable field.
SFMODE[1]	Switches a terminal between screen mode and line mode.
SFNQUE[1]	Accumulates data into a queue for a specified terminal.
SFOPEN[1]	Loads a panel and prepares it for use.
SFPOSR	Establishes a current row in a named table (used only with SFGETI and SFGETR).
SFQTRM[1]	Identifies the current terminal to the Screen Formatting object routines.
SFSETP	Sets the cursor to a selected screen position.
SFSREA[1]	Displays a panel and permits entry of variable values.
SFSWRI[1]	Displays a panel with current variable values.

[1] Indicate routines that are additional or different from the standard screen formatting routines.

CALL Formats

A FORTRAN 5 or COBOL 5 application program calls the screen formatting object routines using the standard subroutine call format for the language used.

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

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

objrtn — The six-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 p1 p2 p3.
```

QTRM Example

Figure 6-1 is a sample FORTRAN QTRM application program, EXAMPLE, which logs in a single user, displays a single screen formatting panel, and logs off the user. This program is functionally similar to the sample program SHOW, described in Displaying Your-Panel, in chapter 3.

Figure 6-2 is a procedure, FOREXAM, that details the NOS system commands necessary to create and execute a QTRM screen formatting application program. You may use this procedure to execute the FORTRAN program in figure 6-1.

Figure 6-3 is an example screen formatting panel (EXAMPAN) that is called by program EXAMPLE. It must be a compiled screen formatting panel resident on local file PANELIB when program EXAMPLE is executed.

```
PROGRAM EXAMPLE(OUTPUT)

***** OVERVIEW.
*
*     *EXAMPLE* IS A QTRM APPLICATION THAT UTILIZES THE
*     QSFLIB OBJECT ROUTINE LIBRARY TO ENABLE USE OF
*     SCREEN FORMATTING WITHIN A DIRECTLY CONNECTED
*     QTRM APPLICATION ENVIRONMENT.
*
****  PROGRAM STRUCTURE.
*
*     EXAMPLE LOGS ON TO THE NETWORK AS LOCAL APPLICATION 1(AP1),
*     AND WAITS FOR A USER TO LOG ON.  WHEN THIS OCCURS, A
*     SCREEN FORMATTING PANEL IS SENT DOWNLINE AND, AFTER VIEWING,
*     THE TERMINATION IS DISCONNECTED AND THE APPLICATION SHUTS DOWN.
*
**    NOTE:  THIS APPLICATION IS FUNCTIONALLY IDENTICAL TO
*     THE SHOW PROGRAM, AS DESCRIBED IN THE SCREEN FORMATTING
*     MANUAL.  THIS EXAMPLE PROGRAM DOES NOT PERFORM EXCEPTION
*     HANDLING AND DOES NOT HANDLE ERROR CONDITIONS.
*
**    THE FOLLOWING QTRM ROUTINES ARE CALLED -
*
*        QTOPEN, QTCLOS, QTENDT, QTPUT, QTGET
*
**    THE FOLLOWING QTRM/SF ROUTINES ARE CALLED -
*
*        SFQTRM, SFMODE, SFOPEN, SFCLOS, SFSWRI, SFDQUE
*


      IMPLICIT INTEGER (A - Z)

      PARAMETER (NUMTERM = 1)
*                                MAX NUMBER OF TERMINAL USERS
      COMMON / NETBUF /  NJUNK (4), NETBUF (100)
*                                NETWORK BUFFER USED FOR I/O
      COMMON / NIT /  NIT (10), TERM (10,NUMTERM)
*                                NETWORK INFORMATION TABLE
      CHARACTER*7 MODEL
*                                TERMINAL MODEL NAME
      CHARACTER*7 PANEL
*                                SCREEN FORMATTING PANEL NAME
      CHARACTER*7  QUESTR
*                                QUEUE NAME
      CHARACTER*27 INSTR, OUTSTR
*                                SCREEN FORMATTING INPUT/OUTPUT STRINGS
      INTEGER BUFF(2500,NUMTERM)
*                                COMMUNICATION BUFFER
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
      INTEGER CURACN
*                            CURRENT CONNECTION NUMBER
      DATA (NIT (I), I = 1, 10) /10 * 0/
      DATA ((TERM (I, J), I = 1, 10), J = 1, NUMTERM) /
     1       NUMTERM (0, 0, 0, 0, 0, 0, 0, 0, 0, 0) /
      DATA QUESTR / 'PUT' /
      DATA PANEL / 'EXAMPAN' /
      DATA MODEL / '721T    ' /
      QTLEN = 8000
      MAXTRAN = 2042
      MAXRL = MAXTRAN * 7.5
      APPNAME = R"AP1     "
      CALL REPLACE ('APPNAME', APPNAME)
      CALL REPLACE ('CHARSET', 11)
      CALL REPLACE ('NUMTERM', NUMTERM)
      CALL REPLACE ('MRL', MAXRL)
      CALL REPLACE ('A2A', 0)
      CALL REPLACE ('SLEEP', -1)
      PRINT*,' '
      PRINT*,' QTRM/SCREEN FORMATTING APPLICATION EXAMPLE'
      PRINT*,' (FUNCTIONALLY IDENTICAL TO *SHOW* PROGRAM)'
      PRINT*,' RUNNING AS LOCAL APPLICATION AP1...'
      CALL QTOPEN (NIT)

*     THE APPLICATION WAITS FOR A USER TO NET ON.

 100  CALL QTGET (NETBUF)
      IF ((EXTRACT('RL').EQ.0) .AND. (EXTRACT('RC').EQ.0)) GO TO 100

      IF (EXTRACT('RC') .EQ. 6) THEN
        PRINT*,' BROKEN CONNECTION '
        STOP
      ELSE IF (EXTRACT('RC').NE. 2) THEN
        PRINT*,' RETURN CODE OTHER THAN 2 - RC = ',EXTRACT('RC')
        STOP
      ELSE
        CURACN = EXTRACT('ACN')
      ENDIF
      CALL REPLACE ('SLEEP', -1)
      CALL REPLACE ('SLEEP',0)
      CALL QTGET (NETBUF)
      CALL REPLACE ('CHARSET', 11)

*     THE APPLICATION WAITS FOR INPUT FROM THE USER.
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
 111  CALL QTGET (NETBUF)
      IF ((EXTRACT('RL').EQ.0) .AND. (EXTRACT('RC').EQ.0)) GO TO 111
      CALL REPLACE ('ACN', CURACN)
      CALL REPLACE ('INTMSG', 0)
      CALL REPLACE ('CHARSET', 11)

      CURACN = EXTRACT ('ACN')

*     SFQTRM - IDENTIFIES THE NIT AND I/O BUFFER TO SFORM.

      CALL SFQTRM (NIT, BUFF(1,CURACN))

*     SFDQUE - QUEUES UP ANY I/O FROM THE SFQTRM CALL.

      CALL SFDQUE (QUESTR, BUFF(1,CURACN), QTRC, QTLEN )
      CALL REPLACE('RL',5)
      CALL SFPUT(BUFF(1,CURACN),EXTRACT('RL'))
*     SFMODE - PLACES THE TERMINAL NAME AND MODEL INTO THE NIT.

      CALL SFMODE (1,MODEL)

*     SFOPEN - OPENS THE PANEL AND PUTS THE TERMINAL INTO SCREEN MODE.

      CALL SFOPEN (PANEL, ISTAT)
      CALL REPLACE ('ACN', CURACN)

      IF (ISTAT .NE. 0) THEN
        PRINT*,' ILLEGAL SFOPEN RETURN STATUS = ',ISTAT
        STOP
      END IF

*     SFSWRI - PLACES THE PANEL OUTPUT STRINGS INTO THE PUT QUEUE.

      CALL SFSWRI (PANEL, INSTR)

*     SFDQUE - DEQUEUES CONTENTS OF PUT QUEUE INTO BUFFER.

      QTLEN = 8000
      CALL SFDQUE (QUESTR, BUFF(1,CURACN), QTRC, QTLEN )

*     SFPUT - SEND PANEL DOWNLINE.

      CALL SFPUT  (BUFF(1,CURACN),EXTRACT('RL') )

*     SFCLOS - CLOSE PANEL AND SEND CLOSING SEQUENCES DOWNLINE.

      CALL SFCLOS(PANEL,1)
      CALL SFDQUE(QUESTR,BUFF(1,CURACN),QTRC,QTLEN)
      CALL REPLACE ('INTMSG', 0)
      CALL SFPUT(BUFF(1,CURACN),EXTRACT('RL'))
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
*     NOTIFY USER OF TERMINATION OF CONNECTION.

      CALL REPLACE ('SLEEP', -1)
      CALL QTGET (NETBUF)

      CALL REPLACE ('INTMSG', 0)

*     TERMINATE USER CONNECTION, NEXT APPLICATION IS IAF.

      CALL REPLACE('NEXTAPP','IAF')
      CALL QTENDT

*     LOG OFF THE APPLICATION FROM THE NETWORK.

      CALL QTCLOSE
      PRINT*,' QTRM/SCREEN FORMATTING SHOW PROGRAM FINISHED.'
      STOP
      END
      SUBROUTINE SFPUT(BFR,LEN)
***
*     ROUTINE SFPUT FORMATS AND SENDS ASCII DATA-TO THE SPECIFIED
*     TERMINAL CONNECTION.

      IMPLICIT INTEGER (A - Z)

      PARAMETER (NUMTERM = 1)
*                            MAX NUMBER OF TERMINAL USERS
      COMMON / NIT /     NIT (10), TERM (10,NUMTERM)
*                            NETWORK INFORMATION TABLE
      COMMON / NETBUF /  NJUNK (4), NETBUF (100)
*                            NETWORK BUFFER USED FOR I/O
      INTEGER BFR(2)
      INTEGER LEN
      INTEGER LCLWRK(1000)

      DO 10 I = 1, 1000
10    LCLWRK(I) = 10H
      TLEN = (LEN+9)/5
      DO 20 I = 1, TLEN
20    LCLWRK(I) = BFR(I)
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
100   CONTINUE
      CALL REPLACE('CHARSET',11)
      CALL REPLACE ('RL',LEN)
      CALL QTPUT (LCLWRK)
      LEN = EXTRACT ('RL')
      CALL REPLACE('SLEEP',0)
      CALL QTGET(NETBUF)
      IF ((EXTRACT('RC').EQ. 5)  .OR.
     1    (EXTRACT('RC').EQ. 12) .OR.
     2    (EXTRACT('RC').EQ. 41))  THEN
        GO TO 100
      ELSE
        RETURN
      ENDIF
      END
          IDENT  QFIELDS
***       QFIELD - MACRO TO DEFINE THE FIELDS IN THE NIT
*
*         THIS MACRO BUILDS FOUR COMMON BLOCKS -
*
*                QNAME  - NAME OF FIELD IN NIT
*                QWORD  - WORD, POSITION AND SIZE OF FIELD IN NIT
*                QCOUNT - STATISTICS WORD TO COUNT THE NUMBER OF
*                         TIMES THAT A FIELD'IS REFERENCED
*                QNUM   - NUMBER OF ENTRIES IN THE ABOVE COMMON BLOCKS
*
NUM       SET    0
QFIELD    MACRO  NAME,GLOBAL,WORD,POS,BITS
          USE    /QNAME/
          VFD    60/10H_NAME
          USE    *
          USE    /QWORD/
          VFD    1/GLOBAL,23/0,12/WORD,12/POS,12/BITS
          USE    *
NUM       SET    NUM+1
          ENDM
          EJECT
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
***       THE FOLLOWING TABLE LISTS THE FIELDS IN THE NIT THAT
*         ARE REFERENCED IN THE ABOVE PROGRAM.
*
*
          QFIELD ACN,1,05,18,12
          QFIELD RL,1,05,36,12
          QFIELD RC,1,05,12,06
          QFIELD CHARSET,1,01,12,06
          QFIELD NEXTAPP,1,06,18,42
          QFIELD APPNAME,1,01,18,42
          QFIELD NUMTERM,1,01,00,12
          QFIELD MRL,1,05,48,12
          QFIELD SLEEP,1,05,30,06
          QFIELD A2A,1,04,00,06
          QFIELD ABL,0,04,54,06
          QFIELD INTMSG,1,05,00,06
          SPACE  5
***       THE FOLLOWING USE AND VFD STATEMENTS MUST RESIDE AFTER
*         THE LAST NIT FIELD DEFINITION TO PLACE THE DIRECTIVE
*         COUNT INTO THE COMMON BLOCK.
*
          USE    /QNUM/
          VFD    60/NUM
          USE    *
          USE    /QCOUNT/
          BSSZ   NUM
          USE    *
          END
      FUNCTION EXTRACT (FIELD)
***
*     FUNCTION EXTRACT RETURNS THE VALUE OF THE SPECIFIED FIELD OF
*     THE NIT.

      IMPLICIT INTEGER (A - Z)

      PARAMETER (NUMTERM = 1)
*                            NUMBER OF TERMINALS
      COMMON /NIT/ NIT (10), TERM (10,NUMTERM)
*                            NETWORK INFORMATION TABLE
      CHARACTER FIELD*10
      COMMON /QNAME/ QNAME (1)
      CHARACTER QNAME*10
      COMMON /QWORD/ QWORD (1)
      COMMON /QCOUNT/ QCOUNT (1)
      COMMON /QNUM/ QNUM
```

Figure 6-1. FORTRAN Example

(Continued)

(Continued)

```
      ACN = SHIFT (NIT (5), 42) .AND. O"7777"
      DO 100 I = 1, QNUM
        IF (FIELD .EQ. QNAME (I)) THEN
          QCOUNT (I) = QCOUNT (I) + 1
          W = SHIFT (QWORD (I), -24) .AND. O"7777"
          S = SHIFT (QWORD (I), -12) .AND. O"7777"
          BITS = QWORD (I) .AND. O"7777"
          M = SHIFT (MASK (BITS), BITS)
          IF (QWORD (I) .LT. 0) ACN = 0
          EXTRACT = SHIFT (TERM (W, ACN), -S) .AND. M
          RETURN
        END IF
 100  CONTINUE
      END

      SUBROUTINE REPLACE (FIELD, VALUE)
***
*     ROUTINE *REPLACE* STORES THE GIVEN VALUE INTO THE SPECIFIED
*     FIELD OF THE NIT.

      IMPLICIT INTEGER (A - Z)

      PARAMETER (NUMTERM = 1)
*                            NUMBER OF TERMINALS
      COMMON /NIT/ NIT (10), TERM (10,NUMTERM)
* .                          NETWORK INFORMATION TABLE
      CHARACTER FIELD*10
      COMMON /QNAME/ QNAME (1)
      CHARACTER QNAME*10
      COMMON /QWORD/ QWORD (1)
      COMMON /QCOUNT/ QCOUNT (1)
      COMMON /QNUM/ QNUM
      ACN = SHIFT (NIT (5), 42) .AND. O"7777"
      DO 100 I = 1, QNUM
        IF (FIELD .EQ. QNAME (I)) THEN
          QCOUNT (I) = QCOUNT (I) + 1
          W = SHIFT (QWORD (I), -24) .AND. O"7777"
          S = SHIFT (QWORD (I), -12) .AND. O"7777"
          BITS = (QWORD (I) .AND. O"7777")
          M = SHIFT (MASK (BITS), BITS + S)
          IF (QWORD (I) .LT. 0) ACN = 0
          TERM (W, ACN) = (TERM (W, ACN) .AND. .NOT. M ) .OR.
     1    ((SHIFT (VALUE, S) .AND. M))
          RETURN
        END IF
 100  CONTINUE
      END
```

Figure 6-1. FORTRAN Example

```
.PROC,FOREXAM.
REWIND,*.
FTN5,I=EXAMPLE,L=LIST.
LOAD(LGO)
LDSET,LIB=QSFLIB/NETIOD,MAP=BSEX/LIST.
NOGO,LGOB.
LGOB.
DAYFILE,L=LIST.
REVERT. QTRM RUN SUCCESSFUL.
EXIT.
REVERT.ABORT. QTRM FAILED.
```

Figure 6-2. Executing Procedure

```
EXAMPAN
{
ATTR DELIMITERS = 'ab' PHYSICAL = ALTERNATE
ATTR DELIMITERS = 'cd' PHYSICAL = BLINK
ATTR DELIMITERS = 'ef' PHYSICAL = INVERSE
ATTR DELIMITERS = 'gh' PHYSICAL = UNDERLINE
ATTR DELIMITERS = 'ij' LOGICAL  = INPUT
ATTR DELIMITERS = 'kl' LOGICAL  = TEXT
ATTR DELIMITERS = 'mn' LOGICAL  = ITALIC
ATTR DELIMITERS = 'op' LOGICAL  = TITLE
ATTR DELIMITERS = 'qr' LOGICAL  = MESSAGE
ATTR DELIMITERS = 'st' LOGICAL  = ERROR
BOX TERMINATOR = '$' WEIGHT = BOLD
 }

         QTRM SCREEN FORMATTING EXAMPLE PANEL.

         THIS PANEL IS AN EXAMPLE TO BE USED WITH
         QTRM/SCREEN FORMATTING APPLICATION *EXAMPLE*.

            THIS PANEL DEMONSTRATES VARIOUS PHYSICAL AND
            LOGICAL ATTRIBUTES AND DIFFERENT TYPES OF BOX
            DRAWING CAPABILITIES.

         $-------------------------------------------$
         |   a1234567890b   PHYSICAL = ALTERNATE     |
         |   c1234567890d   PHYSICAL = BLINK         |
         |   e1234567890f   PHYSICAL = INVERSE       |
         |   g1234567890h   PHYSICAL = UNDERLINE     |
         |   i1234567890j   LOGICAL = INPUT          |
         |   k1234567890l   LOGICAL = TEXT           |
         |   m1234567890n   LOGICAL = ITALIC         |
         |   o1234567890p   LOGICAL = TITLE          |
         |   q1234567890r   LOGICAL = MESSAGE        |
         |   s1234567890t   LOGICAL = ERROR          |
         $-------------------------------------------$
```

Figure 6-3. Screen Panel

SFDQUE(queuename,buffer,rc,length)

This procedure extracts data from queue and places the data into the specified buffer.

SFDQUE returns information in the format returned by QTGET or suitable for QTPUT. SFDQUE is called by an application when it wishes to process a completed input message from the GET queue, or wishes to send a partial or completed message from the PUT queue.

In the case of input, SFDQUE returns the entire input message (specified length should be greater than the largest max-trans-size allowed by the network) in the specified buffer, and updates the status code and network information table (NIT) length field. The application can then process the entire input message as a unit, exactly as if it was received by a call to QTGET. In other words, SFDQUE takes the result of several QTGET/SFNQUE operations and returns a single data block.

For output, SFDQUE returns as much of the queued (from SFNDUE) data as will fit into the specified buffer. The length should be less than or equal to the largest max-trans-size allowed by the network. The application can then QTPUT the contents of the buffer.

If the GET or PUT buffer is emptied by a call to SFDQUE, the buffer space is returned to the system. The NIT field, current-trans-size, is set to the number of characters returned.

The SFDQUE parameters are:

Parameter	Type	Description
queuename	integer	Name of the queue (GET or PUT).
buffer	integer	Address of the data to be added to the queue.
rc	integer	The return code (0 = data queued, 1 = not queued).
length	integer	Size of buffer in 12-bit characters.

SFMODE(mode,model)

The SFMODE object routine switches a terminal between line and screen mode. In addition, SFMODE sets the terminal model name in the NIT.

SFMODE is called by the application each time it is necessary to switch a user between screen and line modes. On the first call to this procedure, a panel control table (PCT) is built for the user. This area is used to hold swap data when control is switched from one user to another.

The SFMODE parameters are:

Parameter	Type	Description
mode	integer	The mode in which to set the user (0 = line, 1 = screen).
model	character	The terminal model name as defined in the TDU definition file.

SFNQUE(queuename,buffer,rc)

This procedure accumulates data into a queue for a specified terminal. Two queues are maintained for each terminal: GET and PUT. A block of 1600 words is allocated with the specified queue name and terminal number.

SFNQUE is called from within the screen formatting routines to build data blocks to be sent to the user's application program. It is also called by the user's application to build a data block for screen formatting. SFNQUE saves information in the format returned by QTGET or suitable for QTPUT.

As implemented for screen formatting, the upper bound limit for the buffer size for building messages is 1000 CM words or 5000 12-bit characters.

The SFNQUE parameters are:

Parameter	Type	Description
queuename	integer	Name of the queue (GET or PUT).
buffer	integer	Address of the data to be added to the queue.
rc	integer	The return code (0 = data queued, 1 = not queued).

SFQTRM(nitaddr,buffer)

The SFQTRM object routine identifies the QTRM NIT, data buffer, and current terminal to the QTRM screen formatting routines.

This is the primary interface between the user's application program and screen formatting. It is called each time a different terminal is being given access to the screen formatting routines. This routine saves all of the screen formatting and terminal information required for a single user (the previous user) in that user's panel control table (PCT).

Once the previous user's information is saved in their PCT, the new (current) user's screen formatting and terminal information is loaded from their PCT into the appropriate areas.

The data saved for each user consists of screen formatting variables and pointers, TDU information, the field data area from the current panel, the panel load table, and the variable data fields from the current panel. The size of the PCT is approximately 300 (decimal) words.

The SFQTRM parameters are:

Parameter	Type	Description
nitaddr	integer	The user's NIT.
buffer	integer	The buffer that the screen formatting routines use to process input/output.

Routines Modified for QTRM Screen Formatting Use

The following routines differ from their standard counterparts to handle the QTRM interface.

SFCLOS(panelname,mode)

This routine does not direct output to the terminal but rather calls SFNQUE to queue the terminal sequences necessary to process the closing of the specified panel.

The current-trans-size field of the NIT contains the number of characters that were queued.

To send these sequences to the current user, a call to SFDQUE must be executed after the SFCLOS routine is called to extract these queued characters from the PUT queue, and place them in a buffer. A QTPUT call is then made with the contents of the buffer, and the sequences are sent downline to the terminal.

SFOPEN(panelname,mode)

This routine does not direct output to the terminal but rather calls SFNQUE to queue the terminal sequences necessary to process the closing of the specified panel.

The current-trans-size field of the NIT contains the number of characters that were queued.

To send these sequences to the current user, a call to SFDQUE must be executed after the SFOPEN routine is called to extract these queued characters from the PUT queue, and place them in a buffer. A QTPUT call is then made with the contents of the buffer, and the sequences are sent downline to the terminal.

SFSWRI(panelname,mode)

This routine does not direct output to the terminal but rather calls SFNQUE to queue the terminal sequences necessary to process the closing of the specified panel.

The current-trans-size field of the NIT contains the number of characters that were queued.

To send these sequences to the current user, a call to SFDQUE must be executed after the SFSWRI routine is called to extract these queued characters from the PUT queue, and place them in a buffer. A QTPUT call is then made with the contents of the buffer, and the sequences are sent downline to the terminal.

SFSREA(panelname,mode)

This routine does not direct output to the terminal but rather calls SFNQUE to queue the terminal sequences necessary to process the closing of the specified panel.

The current-trans-size field of the NIT contains the number of characters that were queued.

To send these sequences to the current user, a call to SFDQUE must be executed after the SFOPEN routine is called to extract these queued characters from the PUT queue, and place them in a buffer. A QTPUT call is then made with the contents of the buffer, and the sequences are sent downline to the terminal.

This routine is modified to preset a return code 23 in the NIT if the SFSREA routine needs more input to finish processing the read operation, and to set the return code equal to zero if the input request is satisfied.

Programming Guidelines

When a QTRM application determines the terminal model and is ready to initiate a full screen dialogue with the current terminal connection, the following sequence of events must occur.

Identifying Connection to QTRM Screen Formatting

To identify the current connection to the QTRM screen formatting routines, a call must be made to the routine SFQTRM. This routine must be called before any other QTRM screen formatting routine. All subsequent calls to QTRM screen formatting routines refer to the terminal that was active at the time of the last SFQTRM call.

Specifying Screen Mode, Terminal Model to QTRM Screen Formatting

The application must next specify the screen mode and terminal model (refer to chapter 3) associated with that particular terminal connection. The terminal model name must be ascertained by the application beforehand from the user. A call to SFMODE is made with the mode specified as screen, and the given terminal model name. This name must correspond to the MODEL_NAME parameter in a TDU terminal definition. This terminal definition may be system defined or may be located on file TERMLIB on user name=LIBRARY.

Opening a Panel

When the application is ready to open a panel and begin formatted screen processing, a call is made to SFOPEN specifying the panel to be opened. The application then checks the status code returned by SFOPEN to verify that the panel was successfully opened and that the specified terminal definition was successfully located.

If the panel was successfully opened, the application may begin a full screen dialogue.

Writing a Panel

Next the application calls SFSWRI, which queues the formatted panel image and necessary terminal control sequences to the PUT queue. The current-trans-size field of the NIT contains the number of characters that were queued. A call to SFSREA must then be processed to have the cursor positioning sequences added to the already queued panel information.

To send the panel data to the current user, a call to SFDQUE must be executed after the SFSWRI routine is called to extract these queued characters from the PUT queue and place them in a buffer. A QTPUT call is then made with the contents of the buffer and the output data is sent downline to the terminal.

Reading from a Panel

Once a panel is written, the application must enable multi-message transparent mode, so the incoming input data is received in a format suitable for processing by the QTRM screen formatting routines. This is accomplished by calling QTTIP to send a supervisory message to enable multi-message transparent mode.

When information is successfully received from the terminal in transparent mode by a QTGET, the application can then queue the incoming data in the GET queue by calling SFNQUE, which stores the data received from the QTGET call.

By calling SFDQUE, the input information can be transferred from the GET queue to the I/O buffer specified on the call to SFQTRM. Routine SFSREA is then called to process this input.

The application checks the return code in the NIT after a call to SFSREA to determine if there is more input required from the user. A return code of 23 indicates there is outstanding input to be processed for that panel, in which case the application must receive more input from the user for that panel. A zero return code specifies that all input was processed correctly and that the read processing is completed.

When all transparent input processing is completed for terminal connection, QTTIP is then called to disable multi-message transparent mode.

Closing a Panel

When the application is ready to close a panel, a call is made to SFCLOS specifying the panel to be closed, and whether the user is to remain in screen mode or line mode.

The current-trans-size field of the NIT contains the number of characters that were queued in the PUT queue that must be sent to the terminal to complete closing the panel and setting the specified mode.

To send the panel data to the current user, a call to SFDQUE must be executed after the SFCLOS routine is called to extract these queued characters from the PUT queue, and place them in a buffer. A QTPUT call is then made with the contents of the buffer, and the output data is sent downline to the terminal.

Appendixes

Code Set Conversion A

The code conversion chart 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. (7-bit ASCII characters occupy the rightmost 7 bits of a 12-bit field. The leftmost 5 bits are unused.) 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 internal 6/12-bit display code used in ASCII mode and the 7-bit ASCII code characters as they appear when displayed in 6-bit display code format. Refer to the NOS Version 2 Reference Set, Volume 3, appendix A, for additional information on character sets and ASCII vs. NORMAL mode.

Table A-1. Code Conversion Chart

ASCII [1] Character	ASCII Octal	ASCII Hexadecimal	6-Bit [2] Character	6-Bit Octal	6/12-Bit [3] Character	6/12-Bit Octal	7-Bit [4] Code 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

1. ASCII refers to the industry-standard, 128-character ASCII character set.

2. 6-Bit refers to the 6-bit display code that supports the CDC graphic 63- or 64-character set.

3. 6/12-Bit refers to the 6/12-bit display code that supports the ASCII graphic 63- or 64-character set.

4. 7-Bit refers to the 7-bit ASCII code, right-justified in 12-bit bytes.

Note: sp represents a space.

(Continued)

Table A-1. Code Conversion Chart *(Continued)*

ASCII [1] Character	ASCII Octal	ASCII Hexa-decimal	6-Bit [2] Char-acter	6-Bit Octal	6/12-Bit [3] Char-acter	6/12-Bit Octal	7-Bit [4] Code Char-acter
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
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	:/

1. ASCII refers to the industry-standard, 128-character ASCII character set.

2. 6-Bit refers to the 6-bit display code that supports the CDC graphic 63- or 64-character set.

3. 6/12-Bit refers to the 6/12-bit display code that supports the ASCII graphic 63- or 64-character set.

4. 7-Bit refers to the 7-bit ASCII code, right-justified in 12-bit bytes.

Note: sp represents a space.

(Continued)

Table A-1. Code Conversion Chart *(Continued)*

ASCII [1] Character	ASCII Octal	ASCII Hexa-decimal	6-Bit [2] Char-acter	6-Bit Octal	6/12-Bit [3] Char-acter	6/12-Bit Octal	7-Bit [4] Code Char-acter
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	:!
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	:;
@ 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

1. ASCII refers to the industry-standard, 128-character ASCII character set.

2. 6-Bit refers to the 6-bit display code that supports the CDC graphic 63- or 64-character set.

3. 6/12-Bit refers to the 6/12-bit display code that supports the ASCII graphic 63- or 64-character set.

4. 7-Bit refers to the 7-bit ASCII code, right-justified in 12-bit bytes.

Note: sp represents a space.

(Continued)

Table A-1. Code Conversion Chart *(Continued)*

ASCII [1] Character	ASCII Octal	ASCII Hexa- decimal	6-Bit [2] Char- acter	6-Bit Octal	6/12- Bit [3] Char- acter	6/12- Bit Octal	7-Bit [4] Code Char- acter
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
` 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*

1. ASCII refers to the industry-standard, 128-character ASCII character set.

2. 6-Bit refers to the 6-bit display code that supports the CDC graphic 63- or 64-character set.

3. 6/12-Bit refers to the 6/12-bit display code that supports the ASCII graphic 63- or 64-character set.

4. 7-Bit refers to the 7-bit ASCII code, right-justified in 12-bit bytes.

Note: sp represents a space.

(Continued)

Table A-1. Code Conversion Chart *(Continued)*

ASCII [1] Character	ASCII Octal	ASCII Hexa-decimal	6-Bit [2] Char-acter	6-Bit Octal	6/12-Bit [3] Char-acter	6/12-Bit Octal	7-Bit [4] Code Char-acter
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;

1. ASCII refers to the industry-standard, 128-character ASCII character set.

2. 6-Bit refers to the 6-bit display code that supports the CDC graphic 63- or 64-character set.

3. 6/12-Bit refers to the 6/12-bit display code that supports the ASCII graphic 63- or 64-character set.

4. 7-Bit refers to the 7-bit ASCII code, right-justified in 12-bit bytes.

Note: sp represents a space.

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.

TDU Syntax Error Mistakes

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 (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.

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.

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.

APPLICATION STRING NAME MUST BE 1 TO 7 CHARACTERS.

Description: The name specified for the application_string was a null string or was greater than 7 characters in length.

Issued by TDU.

User Action: Correct name and resubmit.

CHARACTER VALUE MUST RANGE FROM 0 TO 255.

Description: The value specified for a character in octal, decimal or hexadecimal was not in the range valid for a character specification (0-255). The wrong base may have been used. Decimal is assumed in absence of an explicit base.

Issued by TDU.

User Action: Correct the character value and resubmit.

CONTINUATION EXCEEDS 256 CHARACTERS

Description: The total number of characters in a line and its continuation exceeds 256.

Issued by TDU.

User Action: Reformat using variables and minimize indentation.

CURSOR_BIAS OUT OF RANGE, MUST BE -255 TO 255

Description: Cursor_bias must be within range -255 <= cursor_bias <= 255.

Issued by TDU.

User Action: Correct bias and resubmit.

CURSOR_POS_COLUMN_LENGTH OR CURSOR_POS_ROW_LENGTH MUST BE 0 TO 7

Description: The value specified for the length parameter in question is larger than can be accomodated.

Issued by TDU.

User Action: Correct the length value and resubmit.

DEFINITION FILE NOT FOUND

Description: The user returned the file ZZZZTRM which contains the terminal definitions.

Issued by TDU.

User Action: Re-issue the SCREEN or LINE command with the specified model parameter.

DOUBLY DEFINED PARAMETER xxxxxx

Description: A parameter appeared twice in the same statement.

Issued by TDU.

User Action: Check mixed usage of keyword parameters.

DUPLICATE PARAMETERS, BOTH "IN" AND "INOUT"

Description: Both parameters were specified in the same statement.

Issued by TDU.

User Action: Possible confusion when using parameters. Use "IN" and "OUT" only when the character sequences differ, else use "INOUT".

DUPLICATE PARAMETERS, BOTH "OUT" AND "INOUT"

Description: Both parameters were specified in the same statement.

Issued by TDU.

User Action: Possible confusion when using parameters. Use "IN" and "OUT" only when the character sequences differ, else use "INOUT".

EMPTY INPUT FILE

Description: The input TDL file contained only blank lines (or no lines).

Issued by TDU.

User Action: Check input file.

ERROR ALREADY IN TABLE

Description: A second TABLE declaration was encountered before the TABLEND for the current table.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR ATTRIBUTE NOT LOGICAL

Description: A physical attribute (for example, INVERSE) was specified where a logical attribute (for example, TITLE) was expected.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR ATTRIBUTE NOT PHYSICAL

Description: A logical attribute (for example, TITLE) was specified where a physical attribute (for example, INVERSE) was expected.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR BOX TOO BIG

Description: The specified box has too many corners and/or intersections. There may be no more than 32.

Issued by PDU.

User Action: Redesign box and resubmit.

ERROR CONSTANT WRONG TYPE

Description: The type of constant used was not the type required, for example an INT type constant was used in a RANGE parameter for a REAL variable.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR DIFFERENT NUMBER OF FIELDS THAN DECLARED.

Description: The number of variable fields in the panel image was not the same as the number of VAR statements in the panel declarations.

Issued by PDU.

User Action: Correct the number of declarations and resubmit.

ERROR EMPTY MATCH LIST

Description: A MATCH= parameter was specified with no values in the match list.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EMPTY TABLE

Description: A TABLEND declaration was encountered after a TABLE declaration, with no VAR declarations.

Issued by PDU.

User Action: Remove the TABLE and TABLEND or add VAR(s) and resubmit.

ERROR EXPECTING) AFTER KEY LIST

Description: The key list in a KEY declaration must be terminated by a right parenthesis ( )).

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING (AFTER RANGE=

Description: The range specification in a VAR declaration must be preceded by a left parenthesis (( ).

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING) AFTER RANGE

Description: The range specification in a VAR declaration must be terminated by a right parenthesis ( )).

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING ATTR, BOX, KEY, PANEL, TABLE, VAR or)

Description: Unknown keyword was encountered when the beginning of a new declaration statement or the end of declarations was expected.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING CHAR, INT, OR REAL AFTER TYPE

Description: CHAR, INT and REAL are the only acceptable keywords following TYPE= .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING CONSTANT AFTER =

Description: VAR default value must be a constant.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING CONSTANTS AFTER RANGE

Description: The RANGE parameter value must be two constants enclosed in parentheses.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING DELIMITERS AFTER ATTR

Description: The required parameter DELIMITERS was omitted from an ATTR declaration.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING DELIMITERS, LOGICAL, OR PHYSICAL AFTER ATTR

Description: DELIMITERS, LOGICAL and PHYSICAL are the only acceptable keywords following ATTR .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING END OF BOX DECLARATION

Description: Unknown parameter name within BOX declaration.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR EXPECTING FINE, MEDIUM, OR BOLD AFTER WEIGHT=

Description: FINE, MEDIUM and BOLD are the only acceptable keywords following WEIGHT= .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING IN OR OUT AFTER IO=

Description: IN and OUT are the only acceptable keywords following IO=

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING LIST AFTER MATCH

Description: The MATCH parameter value must be enclosed in parentheses.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING LOGICAL ATTRIBUTE

Description: Unknown keyword following LOGICAL= .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING MUST FILL, ENTER, CONTAIN OR UNKNOWN AFTER ENTRY

Description: MUST FILL, MUST ENTER, MUST CONTAIN and UNKNOWN are the only acceptable keywords following ENTRY= .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING NAME= OR ROWS= AFTER TABLE

Description: Unknown keyword in the TABLE statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING NAME OR TYPE AFTER PANEL

Description: Unknown keyword in the PANEL declaration.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING NORMAL=, ABNORMAL=, HELP= OR MATCH= AFTER KEY

Description: Unknown keyword in the KEY declaration.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING NORMAL=, ABNORMAL=, HELP= OR MATCH=(keys) AFTER KEY

Description: Unknown keyword in the KEY declaration.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING PANEL NAME AFTER PANEL

Description: A panel name is required in the PANEL statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING PHYSICAL ATTRIBUTE

Description: Unknown keyword following PHYSICAL= .

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING PRIMARY OR OVERLAY

Description: PRIMARY or OVERLAY are the only valid panel types.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING QUOTED DELIMITERS

Description: Attribute delimiters must be specified as two characters enclosed in apostrophes; for example, ATTR '()'.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING STRING AFTER HELP

Description: The HELP parameter value must be a character string enclosed in apostrophes; for example, HELP='Helpful message'.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING TABLE DIMENSION

Description: A table dimension (number of time the VARs are to be repeated) must be specified in a TABLE statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING TABLE NAME

Description: A table name must be specified in a TABLE statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING TERMINATOR CHARACTER

Description: A terminator character must be specified in a BOX statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING TERMINATOR= OR WEIGHT= AFTER BOX

Description: Unknown keyword in the BOX statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING TYPE= AFTER PANEL

Description: Unknown keyword in the PANEL statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING VAR, KEY, ATTR, BOX, TABLE OR }

Description: Unknown declaration statement name.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING VAR NAME AFTER VAR

Description: Each variable field declared in a VAR statement must be named.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR EXPECTING X, A, 9, N, E, $, YMD, MDY, OR DMY AFTER FORMAT

Description: An incorrect value was specified for the FORMAT parameter.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR FIELD DECLARED DIFFERENT SIZE

Description: A variable field in the panel image has a different length (number of underlined characters) than declared in the corresponding VAR statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR INPUT FIELD ON FIRST LINE OF IMAGE

Description: An input variable may not be defined on the first line of the image.

Issued by PDU.

User Action: Correct image and resubmit.

ERROR MORE THAN 255 VARIABLES

Description: Only 255 variable fields are allowed per panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 256 BOX ELEMENTS

Description: Only 256 lines and corners are allowed for all box figures in a single panel.

Issued by PDU.

User Action: Reconstruct box figures to conform to limits.

ERROR MORE THAN 256 CONSTANT FIELDS.

Description: There can only be 256 constant fields per panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 32 KEYS

Description: Only 32 function keys may be defined in each panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 511 TOTAL CONSTANT AND VARIABLE FIELDS.

Description: There can be only 511 total fields per panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 64 ATTRIBUTES

Description: Only 64 unique attribute combinations are allowed per panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 8 BOXES

Description: 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.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR MORE THAN 8 TABLES

Description: Only eight TABLE statements are allowed per panel.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR NOT IN TABLE

Description: A TABLEND statement was encountered without a preceding TABLE statement.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR PANEL IMAGE EXCEEDS 64 LINES

Description: A panel may have a maximum of 64 lines.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR RANGE LOW GT HIGH

Description: The constants in a RANGE parameter must have ascending values; for example, the second must be larger than the first.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR RANGE OF CHAR NOT ALLOWED

Description: A CHAR type variable (the default type) cannot have a RANGE parameter.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR REAL CONSTANT FORMAT

Description: 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.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR SFATTR AFTER OTHER ATTRIBUTES

Description: Unimplemented keyword SFATTR was encountered in the panel declaration section.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR SHIFT NOT ALLOWED

Description: SHIFT cannot be specified for the CDC standard keys like BACK. Only the application keys, like F1, can be shifted.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR STRING LENGTH

Description: A single character string within apostrophes exceeds 256 characters.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TABLE DIMENSION REQUIRED

Description: The number of rows in a TABLE must be declared for each table. There is no default.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TABLE NAME REQUIRED

Description: Tables must have a name specified in the TABLE statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TABLE PARAMETER

Description: More than two parameters were specified in the TABLE statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TERMINATOR CHAR REQUIRED

Description: A terminator character enclosed in apostrophes must be specified for each BOX statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TOO MANY ATTR PARAMETERS

Description: More positional parameters than allowed were specified in the ATTR statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TOO MANY VAR PARAMETERS

Description: More positional parameters than allowed were specified in the VAR statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TWO VAR NAMES

Description: More than one name was specified in a single VAR statement. A single VAR can have only one name.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR TYPE/FORMAT MISMATCH IN PRECEDING VAR

Description: The format specified in the VAR statement is not compatible with the data type.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR UNEXPECTED END OF FILE

Description: The end of the panel definition file was encountered before the end of declarations; for example, before the terminating).

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR UNKNOWN KEYWORD

Description: The keyword specified is not allowed for this statement.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR UNTERMINATED STRING

Description: A string with no closing apostrophe was encountered.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR VALIDATION TABLE OVERFLOW

Description: 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.

Issued by PDU.

User Action: 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.

ERROR VALUE TYPE MISMATCH

Description: 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.

Issued by PDU.

User Action: Correct declaration and resubmit.

ERROR VAR DECLARED TWICE

Description: Two VAR statements using the same variable name were encountered. Variable names must be unique.

Issued by PDU.

User Action: Correct declarations and resubmit.

ERROR VAR NAME NOT SPECIFIED

Description: Each VAR statement must specify a variable name.

Issued by PDU.

User Action: Correct declarations and resubmit.

EXPECTING xxxxxx

Description: TDU was expecting to find the indicated symbol, but did not.

Issued by TDU.

User Action: Correct statement and resubmit.

EXPECTING VERB OR VARIABLE, FOUND xxxxxx

Description: A statement began with a symbol other than a name, such as an integer, boolen, string, and so on.

Issued by TDU.

User Action: Correct statement and resubmit.

FUNCTION_KEY_LEAVES_MARK MUST INDICATE 0 TO 7 CHARACTERS OF BLOTCH

Description: TDU cannot accomodate more than 7 characters placed on the screen by a function key.

Issued by TDU.

User Action: Correct the statement and resubmit.

INCORRECT xxxxxx

Description: The indicated symbol is not allowed at the location where it was found.

Issued by TDU.

User Action: Correct statement and resubmit.

INCORRECT COMMUNICATIONS TYPE.

Description: Only SYNCH and ASYNCH are acceptable values for communications type. However, SYNCH is not currently supported.

Issued by TDU.

User Action: Correct statement and resubmit.

INCORRECT CURSOR_ENCODING

Description: Communications value not from allowed set. Must be ASYNCH, SYNCH, or SNA.

Issued by TDU.

User Action: Correct the communication value.

INCORRECT "MOVE_PAST.." OR "CHAR_PAST.." TYPE

Description: 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.

Issued by TDU.

User Action: Use only STOP_NEXT, SCROLL_NEXT, HOME_NEXT, WRAP_ADJACENT_NEXT, or WRAP_SAME_NEXT for the value and resubmit.

INCORRECT NAME--MAY ONLY BE ALPHABETIC AND NUMERIC CHARACTERS

Description: The value assigned to the VALUE parameter of the MODEL_NAME statement used a character other than A-Z or 0-9.

Issued by TDU.

User Action: Use only alphabetic and numeric characters in the name.

INCORRECT TYPE-ONLY STRING, INTEGER, OR VARIABLE ALLOWED

Description: A boolean, undeclared variable, or other symbol was encountered in a character string sequence.

Issued by TDU.

User Action: Check for a misspelled name, missing apostrophe, and so on.

INCORRECT VERB OR MISSING "=" IN VARIABLE ASSIGNMENT

Description: A statement began with a name which TDU did not recognize so it assumed statement was a variable declaration but there was no "=" symbol.

Issued by TDU.

User Action: Check for misspelled statement or missing "=" symbol.

INPUT SEQUENCE FOR xxxxxx IS A DUPLICATE. OF A PREVIOUS ITEM.

Description: The specified input sequence appears in an earlier input specification. It would be impossible to distinguish between the two.

Issued by TDU.

User Action: Change one of the statements and resubmit.

INPUT SEQUENCE FOR xxxxxx IS A SUBSET OF A PREVIOUS ITEM.

Description: The specified input sequence appears earlier as the first characters of a longer sequence. It would be impossible to distinguish between an occurrence of the second sequence and an incomplete occurrence of the earlier sequence.

Issued by TDU.

User Action: Change one of the statements and resubmit.

INPUT SEQUENCE FOR xxxxxx IS A SUPERSET OF A PREVIOUS ITEM.

Description: The leading characters of the specified input sequence duplicate a complete earlier sequence. It would be impossible to distinguish between an occurrence of the earlier sequence and an incomplete occurrence of the second sequence.

Issued by TDU.

User Action: Change one of the statements and resubmit.

INTEGER OVERFLOW xxxxxx

Description: Specified integer exceeds CDC integer size.

Issued by TDU.

User Action: Correct integer value and resubmit.

INTEGER TOO LARGE xxxxxx

Description: Specified integer exceeds CDC integer ceiling.

Issued by TDU.

User Action: Correct integer value and resubmit.

ITEM xxxxxx IS SUPERSET OF A PREVIOUS ITEM

Description: The leading characters of the input character sequence are the same as an entire character sequence encountered earlier.

Issued by TDU.

User Action: All input character sequences must be unique.

NAME IS REQUIRED

Description: The MODEL_NAME statement was missing or the name was invalid.

Issued by TDU.

User Action: You must give your terminal definition file a unique name.

NAME MUST BE 1 TO 6 CHARACTERS

Description: The value assigned to the VALUE parameter of the MODEL_NAME statement used 0 or more than 6 characters.

Issued by TDU.

User Action: Use at least 1 character but no more than 6 characters in the name.

NO ROOM IN TABLE FOR xxxxxx

Description: TDU internal tables exceeded available storage.

Issued by TDU.

User Action: Increase the job's field length limit and retry.

NOT YET IMPLEMENTED xxxxxx

Description: Reserved for future implementation.

Issued by TDU.

User Action: Return to the present

NUMBER OF COLUMNS MUST RANGE FROM 0 to 511

Description: You specified too large a number of columns. It should be within the range 0 to 511.

Issued by TDU.

User Action: Correct number of columns and resubmit.

NUMBER OF ROWS MUST RANGE FROM 0 TO 63

Description: You specified too large a number of rows. It should be within the range 0 to 63.

Issued by TDU.

User Action: Correct number of rows and resubmit.

"OUT" REQUIRED FOR SET_SIZE

Description: The SET_SIZE statement was used without specifying the OUT parameter.

Issued by TDU.

User Action: For every screen size you must specify the character sequence that switches the terminal into that size.

PANEL - ERROR IN xxxxxx CAN'T OPEN FILE yyyyyyy

Description: The specified file containing the panel definitions could not be opened; for example, was not a local file.

Issued by PDU.

User Action: Correct file name or attach, get, or create the definition file.

PANEL - ERROR IN xxxxxx DECLARATIONS

Description: Preceding errors in the declaration part of the panel definition caused compilation to fail. The image is not scanned.

Issued by PDU.

User Action: Correct errors and resubmit.

PANEL - ERROR IN xxxxxx END OF FILE DURING DEFINITIONS

Description: The end of the panel definition file was encountered before the end of declarations; for example, before the terminating).

Issued by PDU.

User Action: Supply missing) or otherwise correct the panel definition and resubmit.

PANEL - ERROR IN xxxxxx NO DEFINITION ON IMAGE

Description: An empty panel definition file was submitted. The definition file must have at least one line.

Issued by PDU.

User Action: Correct definition and resubmit.

PANEL - ERROR IN xxxxxx SCREEN IMAGE

Description: A previously noted error in the panel image caused compilation to fail.

Issued by PDU.

User Action: Correct image and resubmit.

PANEL - ERROR IN xxxxxx UNRECOGNIZED PARAMETER yyy

Description: An unrecognized (probably misspelled) parameter keyword was specified on the PANEL statement.

Issued by PDU.

User Action: Correct parameter specifications and resubmit.

REQUIRED PARAMETER MISSING xxxxxx

Description: The indicated parameter must be specified when this verb is used.

Issued by TDU.

User Action: Supply necessary parameter and resubmit.

SFCLOS PANEL xxxxxxx ALREADY CLOSED

Description: An attempt was made to close a panel more than once.

Issued by SFCLOS.

User Action: Check program logic for a redundant SFCLOS subroutine call for panel xxxxxxx.

SFCLOS PANEL xxxxxxx NOT IN PLT

Description: An attempt was made to close a panel that was never opened.

Issued by SFCLOS.

User Action: Check program logic for a missing SFOPEN subroutine call for panel xxxxxxx.

SFCLOS PANEL xxxxxxx NOT OPENED

Description: An attempt was made to show a panel that was never opened.

Issued by SFSSHO.

User Action: Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSSHO subroutine call.

SFCLOS PANEL xxxxxxx NOT UNLOADED

Description: The fast dynamic loader was unable to unload panel xxxxxxx.

Issued by SFCLOS.

User Action: Call site analyst.

SFOPEN PANEL xxxxxxx BAD ENTRY FORMAT

Description: The passloc/entry list in routine LCP is incorrect.

Issued by SFOPEN.

User Action: Call site analyst.

SFOPEN PANEL xxxxxxx BAD GROUP NAME

Description: The group name of the panel library being used is incorrect.

Issued by SFOPEN.

User Action: Call site analyst.

SFOPEN PANEL xxxxxxx BAD LIBRARY LIST

Description: The library list in routine LCP is incorrect.

Issued by SFOPEN.

User Action: Call site analyst.

SFSREA PANEL xxxxxxx NOT OPENED

Description: An attempt was made to show a panel that was never opened.

Issued by SFSSHO.

User Action: Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSREA subroutine call.

SFSWRI PANEL xxxxxxx NOT OPENED

Description: An attempt was made to show a panel that was never opened.

Issued by SFSWRI.

User Action: Check program to ensure that panel xxxxxxx is successfully opened before it is referenced by an SFSWRI subroutine call.

SFSWRI PANEL xxxxxxx NOT PRIMARY

Description: 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).

Issued by SFSWRI.

User Action: Check program logic to ensure that the primary panel is written on the screen before overlay panels are called.

STRING OVERFLOW xxxxxx

Description: The total number of characters in a string exceeds 256.

Issued by TDU.

User Action: See initialization Verb section. If string is part of a terminal function other than initialization, look for a way to shorten it.

TABLE OVERFLOW xxxxxx

Description: TDU internal tables exceeded available storage.

Issued by TDU.

User Action: Increase the job's field length limit and retry.

TABLE OVERFLOW DURING OPTIMIZATION

Description: TDU internal tables exceeded available storage.

Issued by TDU.

User Action: Increase the job's field length limit and retry.

TDU TERMINATED WITH ERRORS

Description: TDU encountered errors in the terminal definition file as indicated by other messages.

Issued by TDU.

User Action: Correct errors in the input TDL file.

TERMINAL DEFINITION NOT FOUND

Description: There is no file named TERMLIB which contains the specified terminal definitions.

Issued by TDU.

User Action: None.

TERMINAL MODEL NOT YET SPECIFIED

Description: The user has not previously issued a SCREEN or LINE command and so must specify the terminal model name.

Issued by TDU.

User Action: Specify the terminal model name so the SCREEN and LINE commands can be issued without specifying the terminal model.

TOO MANY xxxxxx

Description: A value list was used with a parameter which only allows a single value.

Issued by TDU.

User Action: Correct statement and resubmit.

TOO MANY SCREEN SIZES SPECIFIED, MAXIMUM 4

Description: You specified too many screen sizes.

Issued by TDU.

User Action: Choose your four favorite screen sizes.

UNBALANCED xxxxxx

Description: The indicated symbol should be used in pairs. It was not.

Issued by TDU.

User Action: Check for a missing parenthesis or apostrophe.

UNEXPECTED xxxxxx

Description: TDU did not expect to find the indicated symbol where it did.

Issued by TDU.

User Action: Correct statement and resubmit.

UNKNOWN KEYWORD xxxxxx

Description: TDU did not recognize a parameter.

Issued by TDU.

User Action: Check for misspelling, or extra parenthesis or apostrophe.

VALUE RANGE NOT ALLOWED xxxxxx

Description: TDU does not use value ranges.

Issued by TDU.

User Action: Use a value list.

VARIABLE xxxxxx HAS NOT BEEN DECLARED

Description: The indicated variable was not previously defined.

Issued by TDU.

User Action: Check for misspelling or missing apostrophe.

VERB xxxxxx APPEARS TWICE

Description: Input, Output, and Input/output statements may only appear once.

Issued by TDU.

User Action: Delete the redundant statement.

Glossary C

A

Alternate Intensity Character Display

A CRT display characteristic in which certain characters or character strings are highlighted by displaying them at a light intensity that is different from 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.

C

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 programming business data applications.

D

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 that allows you to make editing changes directly on the permanent copy of the file. Compare with Indirect Access File.

E

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).

F

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 that 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 that 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.

I

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 that allows you to edit a local copy of the file without affecting the permanent copy of the file. When you finish editing the local copy, you can either replace the permanent copy with the local (edited) copy or discard the local copy. Compare 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.

L

Line Mode

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

N

NOS Procedure

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

O

Object Routine

A section of program code that 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.

P

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 that 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.

Q

QTRM

Refer to Queued Terminal Record Manager.

Queued Terminal Record Manager

An interface that provides multi-user full screen capabilities for network applications.

S

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. Compare with Line Mode.

T

Terminal Definition Utility (TDU)

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

U

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.

V

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
```

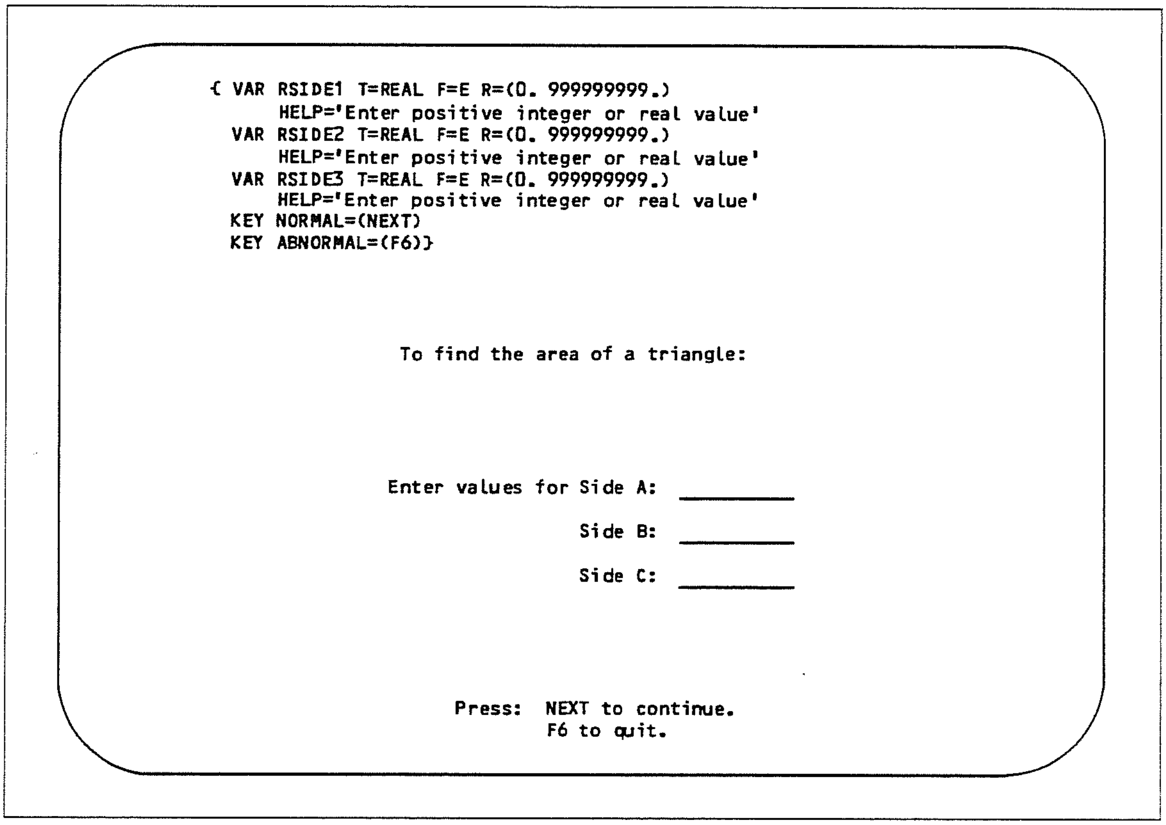
Figure D-1. FORTRAN Program ANGLE3

(Continued)

(Continued)

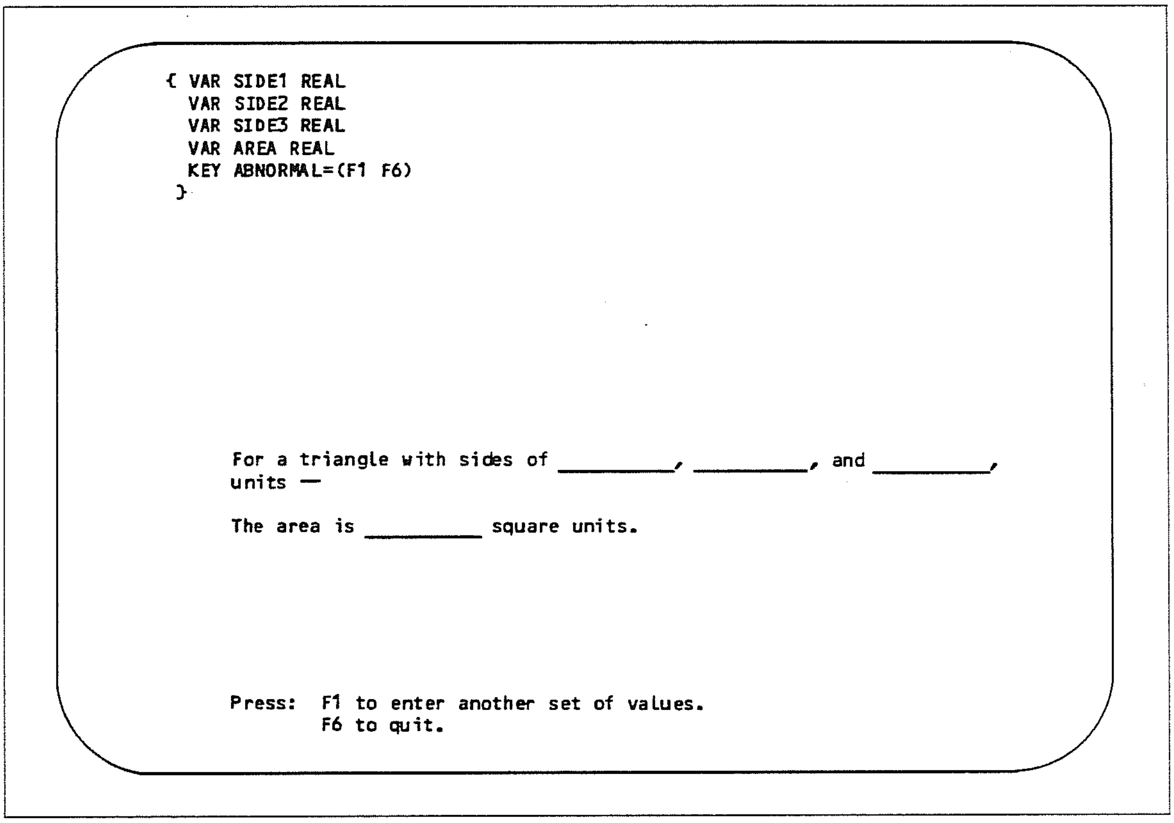
```
C     ***TEST FOR MSGOVL1 SWITCH SETTING. IF SET, CALL
C        BLNKOVL AND CLEAR SWITCH; OTHERWISE, CONTINUE.***
      IF (SW .NE. 0) THEN
         CALL SFSWRI(BLNKOVL,DUMMY)
         SW=0
      ENDIF
C     ***CONVERT INPUT TO REAL VARIABLES***
      READ(INPAN,'(3F10.0)')RSIDE
C     ***CALCULATE 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



```
{ 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.
```

Figure D-7. COBOL Program ESTIMAT

(Continued)

(Continued)

```
*     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.
*
 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).
```

Figure D-7. COBOL Program ESTIMAT

(Continued)

(Continued)

```
       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.
       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.
      *
```

Figure D-7. COBOL Program ESTIMAT

(Continued)

(Continued)

```
      *     THE FOLLOWING SFGETI CALLS RETRIEVE ALL INTEGER VARIABLES
      *          RIGHT JUSTIFIED SO THAT 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.
       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.
      *
```

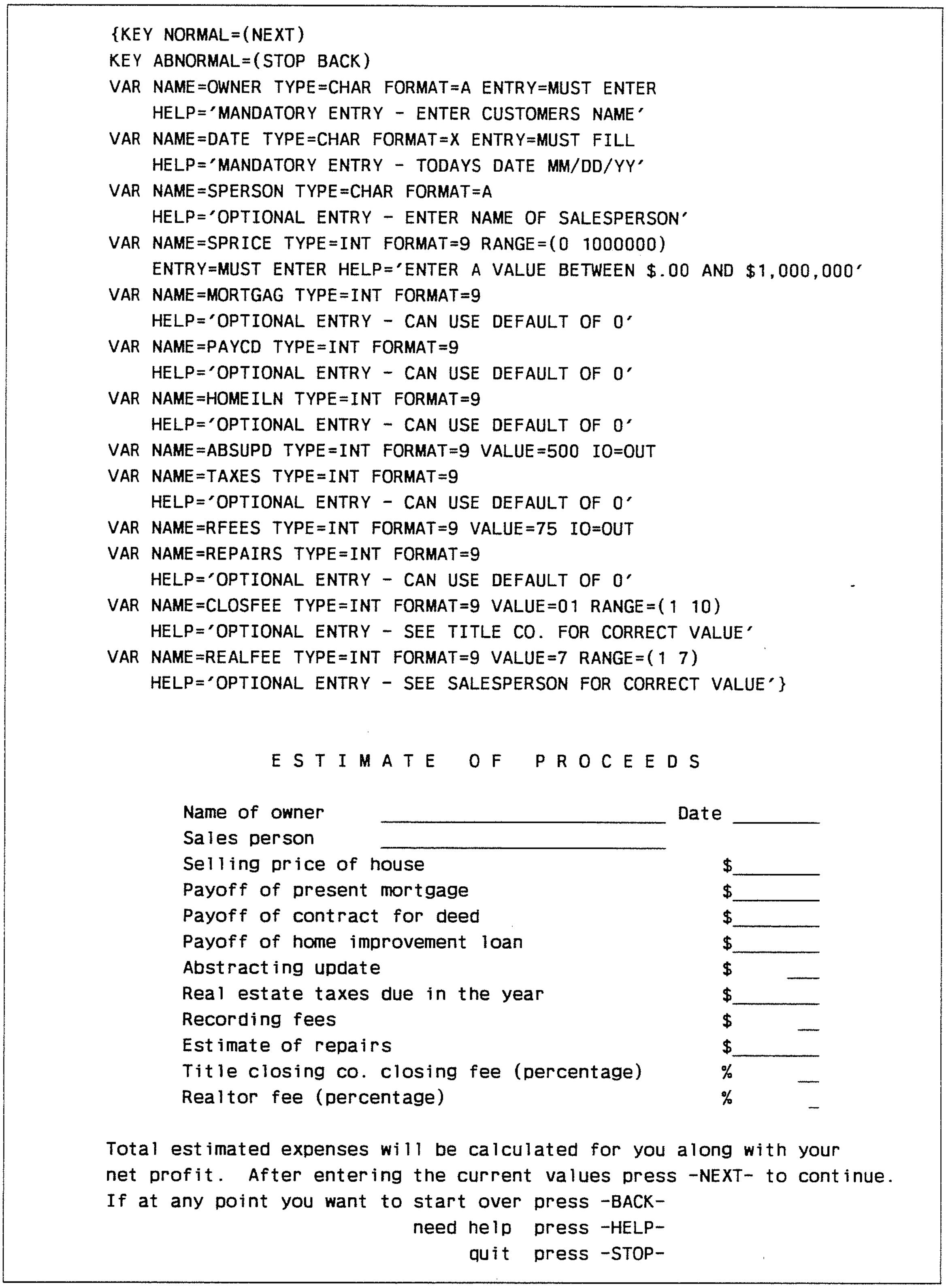
Figure D-7. COBOL Program ESTIMAT

(Continued)

(Continued)

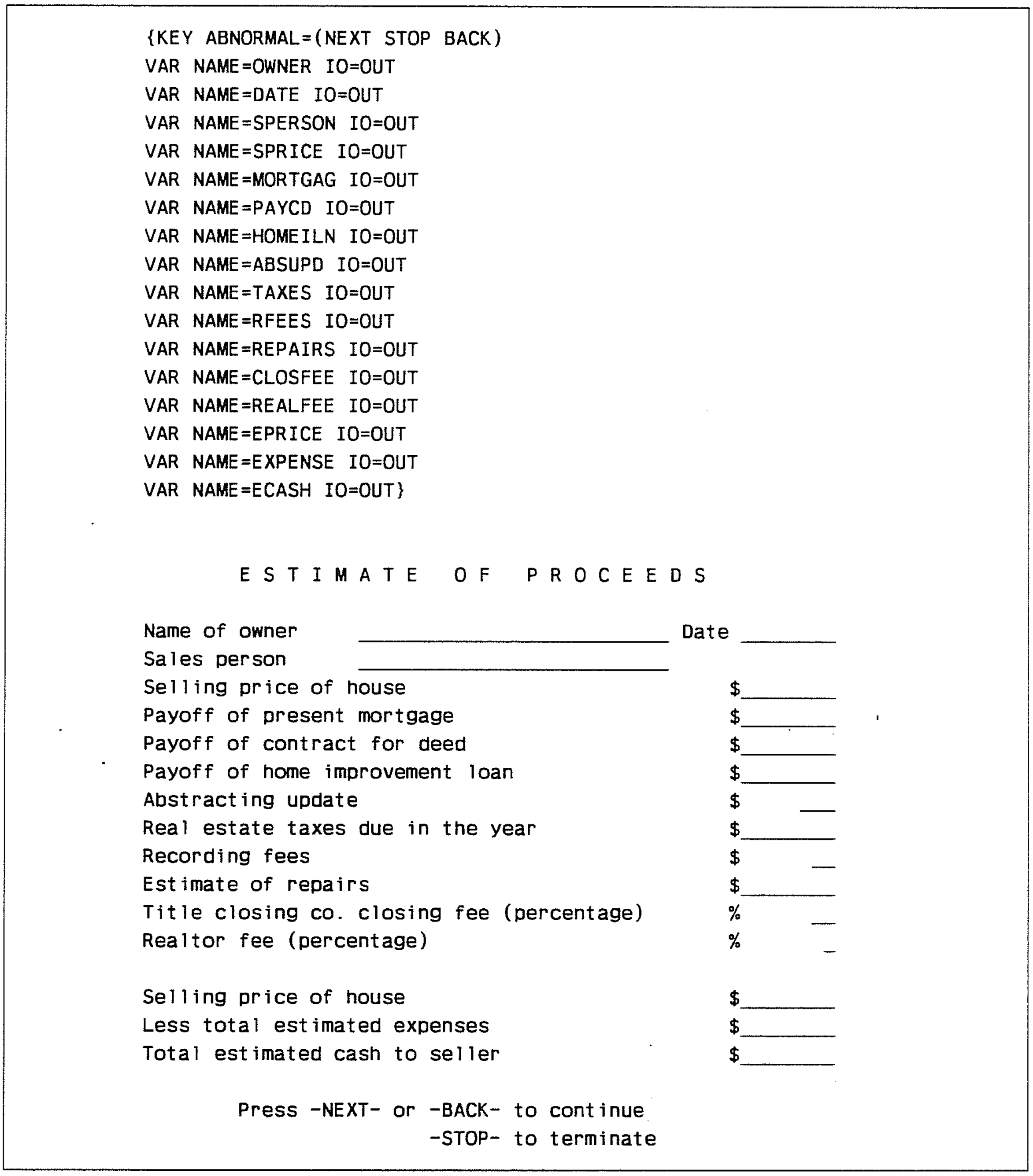
```
*     CHECK HERE FOR -NEXT- OR -BACK- KEYS.
*
     IF NEXT-KEY OR BACK-KEY
         GO TO START-OVER
     END-IF.
*
*     STATUS OF 1 SAYS GO TO LINE MODE.
*
     SET LINE-MODE TO TRUE.
     PERFORM CLOSE-PANELS.
     GO TO STOP-PROGRAM.

 START-OVER.
*
*     STATUS OF 0 SAYS KEEP IN SCREEN MODE.
*
     SET SCREEN-MODE TO TRUE.
     PERFORM CLOSE-PANELS.
     GO TO START-PROGRAM.
 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



```
{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 displays 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 want to avoid the disk access requirements implied by dynamically loading panels. Also, some applications may want 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 is used instead of the default PLT in SFLIB.

Panel Load Table Format

The PLT has a two-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 two-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 that 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    60/1
          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 that 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 that 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 that may include function keys as a subset. The application developer may want 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 must be converted manually.

Optimizations

Programs intended for migration should not static load panels by redefining the default panel load table.

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. The following terminals are system-defined for full-screen use:

- CDC Viking 721 (Version 3 and Version 4 firmware)
- CDC 722
- CDC 722 - 3O[1]
- IBM 3270
- Tektronix 4115[1]
- Zenith Z19/Z29 or Heathkit H19[1]
- DEC VT100[1]
- Lear Siegler ADM3A[1]
- Lear Siegler ADM5[1]

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 chapter 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 some of the other system-defined terminals to generate the same function. If more keys than one 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.

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.

[1] 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 Version 2 Reference Set, Volume 3).

Table G-1. Function Keys on System-Defined Terminals

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				

(Continued)

Table G-1. Function Keys on System-Defined Terminals *(Continued)*

CDC		Lear Siegler	Lear Siegler	Tektronics	Zenith	Digital
Viking 721	CDC 722	ADM3A	ADM5	T4115	Z19	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						

(Continued)

Table G-1. Function Keys on System-Defined Terminals *(Continued)*

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 + NEXT	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					

Index

Index

Comments (continued from other side)

Please fold on dotted line;
seal edges with tape only.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATE

CONTROL DATA
Technology & Publications Division
ARH219
4201 N. Lexington Avenue
Arden Hills, MN 55126-9983

IOS Version 2 Screen Formatting **60460430 D**

/e value your comments on this manual. While writing it, we made some assumptions about who would use and how it would be used. Your comments will help us improve this manual. Please take a few minutes) reply.

/ho are you?

Manager
Systems analyst or programmer
Applications programmer
Operator
Other ______________________

How do you use this manual?

□ As an overview
□ To learn the product or system
□ For comprehensive reference
□ For quick look-up

/hat programming languages do you use? ______________________

____

ow do you like this manual? Check those questions that apply.

'es	Somewhat	No	
□	□	□	Is the manual easy to read (print size, page layout, and so on)?
□	□	□	Is it easy to understand?
□	□	□	Does it tell you what you need to know about the topic?
□	□	□	Is the order of topics logical?
□	□	□	Are there enough examples?
□	□	□	Are the examples helpful? (□ Too simple? □ Too complex?)
□	□	□	Is the technical information accurate?
□	□	□	Can you easily find what you want?
□	□	□	Do the illustrations help you?

omments? If applicable, note page and paragraph. Use other side if needed.

'ould you like a reply? □ Yes □ No

:om:

______________________ ______________________
ame Company

______________________ ______________________
ddress Date

______________________ ______________________
Phone

____

.ease send program listing and output if applicable to your comment.

CONTROL DATA