Terminal Definition for NOS/VE

Usage

CONTROL DATA

60464016

NOS/VE Terminal Definition

Usage

Publication Number 60464016

Manual History

Revision	System Version	PSR Level	Date
A	1.1.2	630	March 1985
B	1.2.1	664	September 1986
C	1.2.2	678	April 1987
D	1.3.1	700	April 1988

Revision D of this manual reflects NOS/VE Version 1.3.1 at PSR level 700. New for this release:

- The INITIALIZE_TERMINAL command.
- Support for terminals with IBM 3270 synchronous communications and block mode operations.
- APPLICATION_STRING statements you can use to maximize system performance.
- Additional terminal definitions.

Contents

Figures

Tables

About This Manual

This manual describes terminal definition procedures for the CONTROL DATA® Network Operating System/Virtual Environment (NOS/VE). The terminal definition statements described in this manual allow you to set up terminals for screen mode applications such as the EDIT_FILE utility.

A Terminal Definition Statements Index follows the last page of this manual. The Index lists all statements alphabetically, along with the page on which each is described.

This manual is part of a set of manuals that describe SCL. If you are not certain this manual includes the information you need, refer to the NOS/VE User Manual Set in this section for abstracts of the other manuals.

Audience

This manual is written for application programmers who want to use existing terminal definition files or create their own. Knowledge of the System Command Language (SCL) as described in the NOS/VE System Usage manual is assumed.

The NOS/VE User Manual Set

This manual is part of a set of user manuals that describe the command interface to NOS/VE. The descriptions of these manuals follow:

Introduction to NOS/VE

Introduces NOS/VE and SCL to users who have no previous experience with them. It describes, in tutorial style, the basic concepts of NOS/VE: creating and using files and catalogs of files, executing and debugging programs, submitting jobs, and getting help online.

The manual describes the conventions followed by all NOS/VE commands and parameters, and lists many of the major commands, products, and utilities available on NOS/VE.

NOS/VE System Usage

Describes the command interface to NOS/VE using the SCL language. It describes the complete SCL language specification, including language elements, expressions, variables, command stream structuring, and procedure creation. It also describes system access, interactive processing, access to online documentation, file and catalog management, job management, tape management, and terminal attributes.

NOS/VE File Editor

Describes the EDIT_FILE utility used to edit NOS/VE files and decks. The manual has basic and advanced chapters describing common uses of the utility, including creating files, copying lines, moving text, editing more than one file at a time, and creating editor procedures. It also contains descriptions of subcommands, functions, and terminals.

NOS/VE Source Code Management

Describes the SOURCE_CODE_UTILITY, a development tool used to organize and maintain libraries of ASCII source code. Topics include deck editing and extraction, conditional text expansion, modification state constraints, and using the EDIT_FILE utility.

NOS/VE Object Code Management

Describes the CREATE_OBJECT_LIBRARY utility used to store and manipulate units of object code within NOS/VE. Program execution is described in detail. Topics include loading a program, program attributes, object files and modules, message module capabilities, code sharing, segment types and binding, ring attributes, and performance options for loading and executing.

NOS/VE Advanced File Management

Describes three file management tools: Sort/Merge, File Management Utility (FMU), and keyed-file utilities. Sort/Merge sorts and merges records; FMU reformats record data; and the keyed-file utilities copy, display, and create keyed files (such as indexed-sequential files).

NOS/VE Terminal Definition

Describes the DEFINE_TERMINAL command and the statements that define terminals for use with full-screen applications (for example, the EDIT_FILE utility).

NOS/VE Commands and Functions

Lists the formats of the commands, functions, and statements described in the NOS/VE user manual set. A format description includes brief explanations of the parameters and an example using the command, function, or statement.

Conventions

The following conventions are used in this manual:

Boldface	In a format, boldface type represents names and required parameters.
Italics	In a format, italic type represents optional parameters.
UPPERCASE	In a format, uppercase letters represent reserved words defined by the system for specific purposes. You must use these words exactly as shown.
lowercase	In a format, lowercase letters represent values you choose.
Blue	In examples of interactive terminal sessions, blue represents user input.
Vertical bar	A vertical bar in the margin indicates a technical change.
Numbers	All numbers are decimal unless otherwise noted.

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. Mail your comments to:

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

Please indicate whether you would like a response.

If you have access to SOLVER, the Control Data online facility for reporting problems, you can use it to submit comments about the manual. When entering your comments, use NV0 (zero) 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 Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

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

Defining Your Terminal 1

Defining Your Terminal 1

Before you can use NOS/VE applications in screen mode you need to specify your terminal's capabilities in a terminal definition. This chapter describes how to:

- Use terminal definitions that are already created and compiled by the system.
- Create a new terminal definition by:
  - Copying and modifying a terminal definition deck found in a source library and then compiling it with the DEFINE_TERMINAL command.
  - Copying and editing a sample deck from a source library and then compiling it with the DEFINE_TERMINAL command.
  - Entering the appropriate terminal definition statements in a text file.
- Download a terminal definition.
- Use terminal definitions in your job.

The terminal definition acts as an interface between screen mode applications, NOS/VE, and your terminal. For everything to work properly, the definition must correctly specify the capabilities of your terminal. Any capability you do not specify is not used. If the definition contains statements that specify the wrong information, either the intended capability will not exist or it will not work properly.

Terminal Capabilities

Your terminal must have certain capabilities to operate in screen mode. These capabilities fall into three categories: required attributes, desired attributes, and optimum attributes.

Required Attributes

To be used in screen mode, your terminal must:

- Use asynchronous or IBM 3270 synchronous communications.
- Operate in character mode or IBM 3270 block mode.
- Have keys that move the cursor on the screen and transmit characters to NOS/VE so that the terminal knows the cursor moved.
- Support direct cursor addressing.
- Provide a screen clear operation.

Desired Attributes

In addition to required attributes, your terminal should also:

- Have a clear-to-end-of-line function.
- Provide at least 16 definable function keys.

Optimum Attributes

Your terminal can achieve optimum performance, if in addition to the required and desired attributes, it also:

- Has up to 32 definable function keys.
- Provides function keys that transmit a unique, identifying character sequence followed by a RETURN character. The RETURN character at the end of function key sequences provides added usability and is a feature of the Control Data 721, the Control Data 722-30, the CONNECT software packages for IBM-compatible PCs and the Apple Macintosh, and other terminals.

  Functions keys on terminals with programmable function keys must be loaded using a unique character sequence that includes the character designated as the RETURN key. The Digital Equipment Corporation VT220 (hereafter referred to as the VT220) is an example of a terminal whose function keys are loaded this way. Refer to appendix D for further information.
- Includes host-definable tab stops for use with the EDIT_FILE utility.
- Includes protected fields on the screen and tabbing between unprotected fields for use with screen formatting. The tab key must transmit characters to the host so that the system knows when the tab key is pressed.
- Has graphic characters for drawing lines.
- Does not use a character position on the screen to enable/disable such attributes as bright, dim, underlined or blinking characters, inverse video, or protected fields.

There are other terminal attributes used by various screen mode applications. However, the first four categories described here are the attributes most frequently used.

Using Existing Compiled Definitions

Each NOS/VE release includes compiled terminal definitions. Your installation probably has the released compiled definitions plus those defined by your site personnel in the file $SYSTEM.TDU.TERMINAL_DEFINITIONS. To get a list of terminal definitions already created and compiled, enter:

```
/display_object_library library=$system.tdu.terminal_definitions
```

Each terminal definition is in a load module in an object library. The load module name is the terminal model name prefixed with CSM$. If your terminal's name is in one of the module names, you can access that module for use with a screen mode application. A list of terminal models for which terminal definitions have been released is included in the Modifying a Terminal Definition Deck section later in this chapter.

Suppose you want to use the EDIT_FILE utility in screen mode at a Zenith Z19 terminal; you would:

1. Check the module list for a name similar to Zenith Z19. Control Data's convention for specifying a model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number, as in ZEN_Z19.

2. Once you locate the model name, which in this example is CSM$ZEN_Z19, enter:

   ```
   /change_terminal_attribute terminal_model=zen_z19
   /change_interaction_style style=screen
   ```

   You can enter these commands after you log in or as part of your user prolog.

An example of a terminal definition for the VT220 terminal is shown in appendix D.

Creating a New Terminal Definition

You can create a new terminal definition by:

- Entering the appropriate terminal definition statements in a text file.
- Copying and modifying terminal definition decks found in a source library and then compiling them with the DEFINE_TERMINAL command.
- Copying and editing a sample deck from a source library and then compiling it with the DEFINE_TERMINAL command.

Entering Terminal Definition Statements

To create a new terminal definition file, enter the appropriate terminal definition statements in a text file and compile the file using the DEFINE_TERMINAL command. (See Compiling a Terminal Definition File later in this chapter.) Each terminal model must be defined on a separate file.

These terminal definition statements are easy to read, but they can be tedious to type. Check to see if someone has already defined your terminal before you create your own file. Also see the next section, which describes how to set up your terminal definition by copying and modifying existing terminal definition decks.

Modifying a Terminal Definition Deck

You can create a new terminal definition by copying and modifying one of the terminal definition decks provided in source library $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE. The terminal models defined in the source library for this release are listed in table 1-1.

To copy the deck you have selected, use the SCU subcommand EXTRACT_DECK. This subcommand produces a source file that you can modify in your catalog.

Example

If you want to create a terminal definition for a Lear Siegler ADM5[1], make a copy of the deck containing the statements for the Zenith Z19 by entering:

```
/scu
sc/use_library base=$system.cybil.osf$program_interface ..
sc../result=$null
sc/extract_deck deck=csm$zen_z19 source=$user.lsi_adm5
sc/quit false
```

A copy of deck CSM$ZEN_Z19 is now on file $USER.LSI_ADM5 ready for modification. After you modify the file, you need to compile it. (See Compiling a Terminal Definition File later in this chapter.)

To modify your copy of any of the decks, refer to the hardware reference manual for your terminal. The manual should list the available keys and attributes and the character sequence your terminal accepts or generates for each key or attribute. You need this information to fill in statement parameters in the file you copy.

Refer to Defining Functions and Key Labels for EDIT_FILE in chapter 2 for information on defining function keys for that utility. Refer also to the NOS/VE File Editor manual.

The example at the end of this chapter shows how to use the CSM$SAMPLE deck to create a new terminal definition.

1. Control Data's convention for specifying a model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number; for example, you might use LSI_ADM5 for the Lear Siegler ADM5.

Table 1-1. Terminal Definitions

Terminal	Deck Name
Apple Macintosh (running CONNECT Version 1.0)	CSM$MAC_CONNECT_10
Apple Macintosh (running CONNECT Version 1.1)	CSM$MAC_CONNECT_11
CDC 721	CSM$CDC_721
CDC 722	CSM$CDC_722
CDC 722-30	CSM$CDC_722_30
CYBER 910-300	CSM$CDC_910
Digital Equipment Corporation VT100	CSM$DEC_VT100 (18 function keys) or CSM$DEC_VT100_GOLD (32 function keys)
Digital Equipment Corporation VT220	CSM$DEC_VT220
IBM 3270	CSM$IBM_3270[1]
IBM 3270 model 2	CSM$IBM_3270_2[1]
IBM 3270 model 3	CSM$IBM_3270_3[1]
IBM 3270 model 4	CSM$IBM_3270_4[1]
IBM 3270 model 5	CSM$IBM_3270_5[1]
IBM PC (running CONNECT Version 1.0)	CSM$PC_CONNECT_10
IBM PC (running CONNECT Version 1.1)	CSM$PC_CONNECT_11
IBM PC (running CONNECT Version 1.2)	CSM$PC_CONNECT_12

1. If you have an Intercom network, this terminal definition is not supported.

(Continued)

Table 1-1. Terminal Definitions *(Continued)*

Terminal	Deck Name
IBM PC (running CONNECT Version 1.3)	CSM$PC_CONNECT_13
Sun Microsystems 3/160	CSM$SUN_160
Tektronix 4109	CSM$TEK_4109
Tektronix 4115	CSM$TEK_4115
Tektronix 4125	CSM$TEK_4125
TeleVideo 950	CSM$TV_950[2]
TeleVideo 955	CSM$TV_955[2]
TeleVideo 950	CSM$TV_950_ PROTECTED[3]
TeleVideo 955	CSM$TV_955_ PROTECTED[3]
Zenith Z19 or Heathkit H19	CSM$ZEN_Z19
Zenith Z29 or Heathkit H29	CSM$ZEN_Z29

2. This terminal definition defines the *insert* and *delete* keys. If you use the EDIT_FILE utility often, you will probably need these keys. This definition does not provide automatic positioning of the cursor when filling in fields in a screen mode form.

3. This terminal definition makes filling in fields on forms easier to do. After you fill in a field on a form, the cursor automatically positions to the next field. This terminal definition does not define the *insert* and *delete* keys, which are of use in the EDIT_FILE utility.

Modifying the Sample Deck

To create a new file, copy and edit deck CSM$SAMPLE in source library $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE. This deck contains all the terminal definition statements, formatted correctly, with directions for filling in the parameters to describe your terminal.

If you copy deck CSM$SAMPLE, carefully follow the directions for filling in statement parameters (the directions are enclosed in quotation marks before each statement). Deck CSM$SAMPLE lists statements for all possible attributes and keys that can be supported by screen mode applications. Not all attribute and key statements may apply to your terminal. Leave those that do not apply blank. (Decks other than CSM$SAMPLE contain only those statements needed to define the specified terminal.)

Refer to Defining Functions and Key Labels for EDIT_FILE in chapter 2 for information on defining function keys for that utility. Refer also to the NOS/VE File Editor manual.

The example at the end of this chapter shows how to use the CSM$SAMPLE deck to create a new terminal definition.

Compiling a Terminal Definition File

The DEFINE_TERMINAL command compiles your terminal definition, and creates an object library file of terminal definition modules that can be used by the EDIT_FILE utility and other screen mode applications. Subsequent executions of DEFINE_TERMINAL will merge the new terminal definition with previously compiled definitions (assuming you use the same object library file.) Therefore, one object library can contain all your compiled terminal definitions, even though each definition originates from its own file.

DEFINE_TERMINAL Command Format

The format of the DEFINE_TERMINAL command is:

> **DEFINE_TERMINAL** or **DEFT**
> **INPUT=file**
> *BINARY=file*
> *LIST=file*

The INPUT (I) parameter specifies the terminal definition file you want to compile. Each input file can contain only one terminal definition. This parameter is required.

The BINARY (B) parameter specifies the object library file that is to contain the compiled module (the description of the object library precedes the command format). If you omit the BINARY parameter, object library TERMINAL_DEFINITIONS under your working catalog is assumed.

The LIST (L) parameter specifies the file you want to contain intermediate output from the compilation process (CYBIL code). Most users do not need to see this file. If omitted, $LIST is assumed.

Object Library Characteristics

In the object library file or in the terminal definition file, the terminal definition module is identified by the name you enter on the MODEL_NAME statement. You enter the name on the VALUE parameter prefixed with the characters CSM$. If a module with the same name is already in the object library, the new module replaces the one in the library.

To delete modules from the object library, use the CREATE_OBJECT_LIBRARY subcommand DELETE_MODULE (refer to the NOS/VE Object Code Management manual).

To save your terminal definition, keep your object library on a permanent file.

Example

If you want to set up your own terminal definition for the Lear Siegler ADM5[2] terminal, copy sample deck CSM$SAMPLE from source library $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE to your own file by entering:

```
/scu
sc/use_library base=$system.cybil.osf$program_interface ..
sc../result=$null
sc/extract_deck deck=csm$sample source=lsi_adm5
sc/quit false
```

2. Control Data's convention for specifying a model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number; in this example, the Lear Siegler ADM5 is abbreviated LSI_ADM5.

Edit the source file with the correct information for the Lear Siegler ADM5 terminal, using the following model name:

```
model_name value='lsi_adm5'
```

Compile the file as follows:

```
/define_terminal input=lsi_adm5..
../binary=$user.terminal_definitions
```

The terminal definition for the Lear Siegler ADM5 terminal is merged into file $USER.TERMINAL_DEFINITIONS. The entry in the DISPLAY_OBJECT_LIBRARY listing of $USER.TERMINAL_DEFINITIONS is CSM$LSI_ADM5 and the model name is LSI_ADM5.

Downloading a Terminal Definition

If you have access to SOLVER (Control Data's online database for reporting problems), you can download terminal definition files set up for terminals other than the one you are using. These files are under the special product code CSF. You can use either the XMODEM or CONNECT/RMF protocols to download the files with a microcomputer.

A one-line description for each file describes the terminal definition contained in that file. Enter the SOLVER search request

```
product=csf,text=o
```

to produce a short description of each terminal definition file that is available.

Please have your site analyst contact CDC CYBER Software Support if you want to place any locally developed terminal definitions on SOLVER for other sites to access.

Using Your Terminal Definition

To use your own terminal definition for a screen mode application, you must add the library containing your terminal definitions to your job library list. This is done with the SET_PROGRAM_ATTRIBUTE command. The format for adding libraries is:

SET_PROGRAM_ATTRIBUTE or **SETPA**
ADD_LIBRARY=list of file

The complete format is described in the NOS/VE Object Code Management manual.

Example:

To add object library TERMINAL_DEFINITIONS to your job library list, enter:

```
/set_program_attribute add_library=$user.terminal_definitions
```

To set up your own terminal definition for the Lear Siegler ADM5,[3] enter:

```
/change_terminal_attributes terminal_model=lsi_adm5
/change_interaction_style style=screen
```

You may want to add the SET_PROGRAM_ATTRIBUTE, CHANGE_TERMINAL_ATTRIBUTES, and CHANGE_INTERACTION_STYLE commands to your user prolog. Then, each time you log in, your library of terminal definitions will be added to the job library list automatically as will your terminal model and interaction style.

If you want to share your definitions with others at your site, either make your object library public and have others add it to their job library list or talk to site personnel about adding your definitions to the $SYSTEM.TDU.TERMINAL_DEFINITIONS file.

3. Control Data's convention for specifying a model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number; in this example, the Lear Siegler ADM5 is abbreviated LSI_ADM5.

Terminal Definition Statements 2

Terminal Definition Statements 2

Terminal definition statements describe the capabilities of a specific terminal and the system with which it interacts.

This chapter:

- Describes the general format of terminal definition statements.
- Lists the statements required for proper functioning of any screen mode application.
- Lists and describes all supported terminal definition statements.

The statement types:

- Attribute statements
- Cursor positioning information statements
- Cursor behavior statements
- Screen size specification statements
- Statements that define functions and labels for applications
- A command that initializes the terminal for line or screen mode interaction style
- Screen mode application statements
- Input/output statements
- Input statements
- Output statements

General Format of Terminal Definition Statements

The format of a terminal definition statement conforms to the SCL naming conventions with the following exceptions.

- The statement name BACKSPACE has no abbreviation.
- The parameter name INOUT is abbreviated to IO, rather than I, to distinguish it from the abbreviation for IN.
- The OUT parameter of the APPLICATION_STRING statement can be continued on more than one line under the following conditions:
  - Strings that would extend over more than one line must be broken into substrings that the system concatenates. Each substring must be complete on a single line.
  - Variables must be complete on each line.

Refer to the NOS/VE System Usage manual for more information about naming conventions.

All terminal definition statements have the same general format.

```
statement_name or
abbreviated_statement_name
    parameter name=value list
    parameter name=value list
      :
    parameter name=value list
```

Most frequently, value list is a character string you can find in the hardware reference manual for your terminal. Often these tables represent a character in different ways; for example:

Representation	**Meaning**
'A'	The ASCII character A. Enter printable ASCII characters as strings. (See appendix C for a complete list of ASCII characters.)
101(8)	The character A as an octal number.
41(16)	The character A as a hexadecimal number.

Representation	Meaning
65	The character A as a decimal number.
33(8)	The ASCII ESC character as an octal number.
ESC	The ASCII ESC character indicated by its standard designation. Enter nonprintable ASCII characters as keywords. (See the Graphic or Mnemonic column of table C-1 in appendix C for a list of standard designations for ASCII characters).

When you have more than one item in the value list, put the list in parentheses with each item separated by a blank or comma.

Example:

The following are valid terminal definition statements.

```
model_name value='CDC721'
blink_begin out=(esc 12(16) 'a')
```

These statements show values in different ways:

- As ASCII character strings:

  'CDC721' and 'a'

- As an ASCII character mnemonic:

  ESC

- As a hexadecimal number:

  12(16)

If you intend to use a character string more than once, you may want to define a variable name to have the value of that string. You can do this by equating the variable name to its value at the beginning of the terminal definition, before any of the statements. The format is:

variable_name=list of character string

The variable name can be any string of alphanumeric characters and the underscore, beginning with an alphabetic character. It can be up to 256 characters. The value of the string is the sequence listed in your terminal hardware reference manual for a particular attribute. The separator between each item in list of character string can be either a comma or one or more spaces.

Example:

Assume that the hardware reference manual for your terminal specifies the following sequence be used to enable a protected field:

rs dc2 'K'

You then define a variable name to have that value by entering

```
enable_protect=(rs dc2 'K')
```

at the beginning of the terminal definition. Throughout the remainder of the definition, you then use ENABLE_PROTECT as a value in place of the character string.

Required Terminal Definition Statements

Some statements are required in order for full screen applications to work correctly. These are:

CHAR_PAST_LAST_POSITION
CHAR_PAST_LEFT
CHAR_PAST_RIGHT
COMMUNICATIONS
CURSOR_DOWN

CURSOR_LEFT
CURSOR_POS_BEGIN
CURSOR_POS_ENCODING
CURSOR_POS_SECOND (if applicable)

CURSOR_POS_THIRD (if applicable)
CURSOR_RIGHT
CURSOR_UP
ERASE_PAGE_STAY or ERASE_PAGE_HOME
FUNCTION_KEY_LEAVES_MARK

MODEL_NAME or TERMINAL_MODEL
MOVE_PAST_BOTTOM
MOVE_PAST_LEFT
MOVE_PAST_RIGHT
MOVE_PAST_TOP

There must also be a subset of the application function keys available and defined (a minimum of 16).

The ERASE_END_OF_LINE statement is not required, but it is highly recommended.

NOTE

In the brief descriptions later in this chapter, all required statements are in bold type. Also, the format description of each required statement states that it is required.

Attribute Statements

Overview

Attribute statements describe or determine general characteristics of the terminal. A brief description of each attribute follows. Required statements are in boldface type. (See the next section for statement formats and detailed descriptions.)

Statement	Description
AUTOMATIC_TABBING	Indicates whether the terminal supports tabbing from one completely filled, unprotected input field to the next, without requiring that a tab key be pressed.
CLEARS_WHEN_CHANGE_SIZE	Determines whether the screen clears when the screen size changes.
COMMUNICATIONS	Identifies the type of terminal communication. The only type of communication supported is asynchronous (TYPE=ASYNCH). This statement is required.
FIXED_TAB_POSITIONS	Indicates the position of the fixed tab stops.
FUNCTION_KEY_LEAVES_MARK	Specifies the number of characters that must be repainted when you press a function key. This statement is required.

Statement	Description
HAS_HIDDEN	Allows you to define areas on the screen in which something typed will not be displayed.
HAS_PROTECT	Allows you to use the PROTECT_ BEGIN and PROTECT_END statements to define protected areas on the screen.
HOME_AT_TOP	Determines where the CURSOR_ HOME statement sends the cursor, to the top left of the screen or to the bottom.
MODEL_NAME	Identifies the type of terminal being defined. Either this statement or the TERMINAL_MODEL statement is required.
MULTIPLE_SIZES	Indicates whether your terminal supports more than one screen size.
PROGRAMMABLE_TAB_ STOPS	Identifies the number of programmable tab stops.
TABS_TO_HOME	Determines whether the TAB key moves the cursor to the cursor home position or wraps around to the first unprotected field when the cursor is at the last unprotected field.

Statement	Description
TABS_TO_TAB_STOPS	Specifies whether the terminal supports tabbing to settable or predefined tab stops (like typewriter tabs).
TABS_TO_UNPROTECTED	Specifies whether the terminal supports tabbing forward and backward to the start of unprotected fields.
TERMINAL_MODEL	Identifies the type of terminal being defined. Either this statement or the MODEL_NAME statement is required.
TYPE_AHEAD	Allows a full screen application to execute in type ahead mode.

Format Descriptions

All attribute statements except COMMUNICATIONS and PROGRAMMABLE_TAB_STOPS have a VALUE parameter. This parameter is used in different ways depending on the statement (refer to individual descriptions for complete information).

AUTOMATIC_TABBING

The AUTOMATIC_TABBING statement indicates whether the terminal supports tabbing from one completely filled, unprotected input field to the next, without requiring that a tab key be pressed. If omitted, it is assumed that the terminal does not have this capability.

The format is:

```
AUTOMATIC_TABBING or AUTT
   VALUE=boolean
```

The VALUE (V) parameter indicates whether the terminal supports tabbing from one input field to the next. Specify TRUE if the terminal supports both tabbing and protected areas. Specify FALSE if it does not support both tabbing and protected areas. This parameter is required.

CLEARS_WHEN_CHANGE_SIZE

The CLEARS_WHEN_CHANGE_SIZE statement determines whether the screen clears when the screen size changes. If omitted, the screen does not clear.

The format is:

```
CLEARS_WHEN_CHANGE_SIZE or CLEWCS
   VALUE=boolean
```

The VALUE (V) parameter determines whether the screen clears. Specify TRUE to clear the screen. Specify FALSE if you do not want the screen to clear, or if your terminal supports only one screen size. This parameter is required.

COMMUNICATIONS

The COMMUNICATIONS statement identifies the type of communication your terminal uses. This statement is required.

The format is:

COMMUNICATIONS or **COM**
  TYPE=keyword

The TYPE (T) parameter identifies the terminal protocol. Specify ASYNCH, SYNCH, or SNA. This parameter is required.

FIXED_TAB_POSITIONS

The FIXED_TAB_POSITIONS statement identifies the locations of the fixed tab positions on the terminal.

The format is:

FIXED_TAB_POSITIONS or **FIXTP**
  POSITIONS=list of integer

The POSITIONS (P) parameter specifies the tab positions (list of integers) that are set for the terminal. This parameter is required.

FUNCTION_KEY_LEAVES_MARK

The FUNCTION_KEY_LEAVES_MARK statement is needed for full screen products to repaint the valid characters after a function key press. Use this statement if the following applies:

- Pressing a function key causes characters to appear on the screen.
- Function keys require escape or control sequences that include a character to complete the sequence.

This statement is required.

The format is:

FUNCTION_KEY_LEAVES_MARK or **FUNKLM**
VALUE=integer

The VALUE (V) parameter specifies the number of characters that must be erased from the screen (in order for the original characters to be repainted) after a function key is pressed. If your terminal does not write characters when a function key is pressed, enter a value of 0. This parameter is required.

HAS_HIDDEN

The HAS_HIDDEN statement allows you to use the HIDDEN_BEGIN and HIDDEN_END statements. If your terminal has the capability, these statements define areas on the screen in which something typed will not be displayed. If the statement is omitted, no hidden areas can be defined.

The format is:

HAS_HIDDEN or **HASH**
VALUE=boolean

The VALUE (V) parameter specifies whether the HIDDEN_BEGIN and HIDDEN_END statements can be used. Specify TRUE if your terminal is capable of having areas hidden. Specify FALSE if the capability does not exist on your terminal or if the terminal uses a character position on the screen to provide this capability. This parameter is required.

HAS_PROTECT

The HAS_PROTECT statement allows you to use the PROTECT_BEGIN and PROTECT_END statements. If your terminal has the capability, these statements define protected areas on the screen. If omitted, no protected areas can be defined.

The format is:

HAS_PROTECT or **HASP**
VALUE = boolean

The VALUE (V) parameter specifies whether the PROTECT_BEGIN and PROTECT_END statements can be used. Specify TRUE if your terminal is capable of having areas protected. Specify FALSE if the capability does not exist on your terminal or if the terminal uses a character position on the screen to provide this capability. This parameter is required.

HOME_AT_TOP

The HOME_AT_TOP statement determines whether the CURSOR_HOME statement sends the cursor to the top left of the screen or to the bottom. To ensure the proper functioning of the EDIT_FILE utility, include this statement with VALUE=TRUE. If omitted, the cursor home position is at the bottom left of the screen.

The format is:

HOME_AT_TOP or **HOMAT**
VALUE = boolean

The VALUE (V) parameter determines the home position of the cursor. Specify TRUE for the cursor home position to be at the top left of the screen. Specify FALSE for the cursor home position to be at the bottom left of the screen. This parameter is required.

MODEL_NAME

The MODEL_NAME statement identifies the type of terminal being defined. This statement is required.

The format is:

MODEL_NAME or **MODN**
  VALUE = string

The VALUE (V) parameter specifies the model name to be used:

- As the TERMINAL_MODEL on the CHANGE_TERMINAL_ATTRIBUTES command.
- On the subcommand that activates screen mode for an application.
- As the name of the compiled terminal definition file on an object library (the model name is prefixed by CSM$).

The terminal model name you specify for the VALUE parameter is a string that consists of 1 through 25 alphanumeric characters and the underscore character, and starts with an alphabetic character. The system does not distinguish between uppercase and lowercase characters. CDC_721 and cdc_721 are both interpreted as CDC_721. Control Data's convention for specifying a model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number; for example, DEC_VT100.

The VALUE parameter is required.

MULTIPLE_SIZES

The MULTIPLE_SIZES statement specifies whether your terminal can support more than one screen size. You must include the MULTIPLE_SIZES statement with the SET_SIZE statement. (See the Screen Size Specification section later in this chapter.)

The format is:

MULTIPLE_SIZES or **MULS**
  VALUE = boolean

The VALUE (V) parameter specifies whether more than one SET_SIZE statement can be used. If your terminal can have more than one screen size within a screen mode application, specify TRUE. If it can have only one screen size, specify FALSE. This parameter is required.

PROGRAMMABLE_TAB_STOPS

The PROGRAMMABLE_TAB_STOPS statement identifies the number of programmable tab stops.

The format is:

> **PROGRAMMABLE_TAB_STOPS** or **PROTS**
> **NUMBER=integer**

The NUMBER (N) parameter identifies the number of programmable tab stops. This parameter is required.

TABS_TO_HOME

The TABS_TO_HOME statement determines whether the TAB key moves the cursor to the cursor home position or wraps around to the first unprotected field, when the cursor is at the last unprotected field. (The reverse happens when you tab backward.) If omitted, the TAB key tabs to the first unprotected field.

The format is:

> **TABS_TO_HOME** or **TABTH**
> **VALUE=boolean**

The VALUE (V) parameter determines whether the TAB key moves the cursor to the cursor home position or wraps around to the first unprotected field. Specify TRUE if you want the cursor to go to the home position. Specify FALSE if you want the cursor to wrap around to the first unprotected field, or if the terminal does not have protected areas. This parameter is required.

TABS_TO_TAB_STOPS

The TABS_TO_TAB_STOPS statement specifies whether the terminal supports tabbing to settable or predefined tab stops (like typewriter tabs). If omitted, it is assumed the terminal does not have tab stops.

The format is:

TABS_TO_TAB_STOPS or **TABTTS**
VALUE=boolean

The VALUE (V) parameter specifies whether the terminal has tab stops. Specify TRUE if the terminal has tab stops. Specify FALSE if it does not have tab stops. This parameter is required.

TABS_TO_UNPROTECTED

The TABS_TO_UNPROTECTED statement specifies whether the terminal supports tabbing forward and backward to the start of unprotected fields. If omitted, it is assumed the terminal does not support this type of tabbing.

The format is:

TABS_TO_UNPROTECTED or **TABTU**
VALUE=boolean

The VALUE (V) parameter specifies whether the terminal supports tabbing forward and backward to the start of unprotected fields. Specify TRUE if the terminal supports this type of tabbing. Specify FALSE if the terminal does not support it or if the terminal does not have protected areas. This parameter is required.

TERMINAL_MODEL

The TERMINAL_MODEL statement identifies the type of terminal being defined. Either this statement or the MODEL_NAME statement is required.

The format is:

TERMINAL_MODEL or **TERM**
  VALUE = string

The VALUE (V) parameter specifies the terminal model name to be used:

- As the TERMINAL_MODEL on the CHANGE_TERMINAL_ATTRIBUTES command.
- On the subcommand that activates screen mode in a screen mode application.
- As the name of the compiled terminal definition file on an object library (the model name is prefixed by CSM$).

The terminal model name you specify for the VALUE parameter is a string that consists of 1 through 25 alphanumeric characters and the underscore character; it must begin with an alphabetic character. The system does not distinguish between uppercase and lowercase characters. CDC_721 and cdc_721 are both interpreted as CDC_721. Control Data's convention for specifying the model name is to use a three-character abbreviation for the terminal manufacturer's name followed by the model number; for example, DEC_VT100.

The VALUE parameter is required.

TYPE_AHEAD

The TYPE_AHEAD statement allows a screen mode application to execute in type ahead mode. In type ahead mode you can enter additional input without waiting for the system to respond to previous input. This statement is included for compatibility with NOS terminal definitions. NOS/VE executes applications in type ahead mode no matter what you specify here. If omitted, type ahead mode is assumed.

The format is:

TYPE_AHEAD or **TYPA**
  VALUE = boolean

The VALUE (V) parameter specifies type ahead mode. Enter either TRUE or FALSE. This parameter is required.

Cursor Position Information Statements

The cursor position information statements define the terminal attributes of the cursor position. A brief description of each statement follows. Required statements are in boldface type. (See the next section for statement formats and detailed descriptions.)

Statement	Description
CURSOR_POS_BEGIN	Specifies the first character string of the cursor position sequence. This statement is required.
CURSOR_POS_COLUMN_FIRST	Indicates the column versus row cursor position sequence.
CURSOR_POS_COLUMN_LENGTH	For ANSI type terminals, indicates the number of bytes your terminal sends for column values.
CURSOR_POS_ENCODING	Indicates how your terminal encodes the cursor position output sequence. This statement is required.
CURSOR_POS_ROW_LENGTH	For ANSI type terminals, indicates the number of bytes your terminal sends for row values.
CURSOR_POS_SECOND	Specifies the second character string of the cursor position sequence. This is a required statement if applicable to your terminal.
CURSOR_POS_THIRD	Specifies the third character string of the cursor position sequence. This is a required statement if applicable to your terminal.

Format Descriptions

Each cursor position information statement description follows.

CURSOR_POS_BEGIN

The CURSOR_POS_BEGIN statement specifies the first character to which the cursor is positioned. For example, in the encoding sequence axbyc, the first character the cursor is positioned to is a. (The description of the CURSOR_POS_ENCODING statement later in this chapter provides more information).

The CURSOR_POS_BEGIN statement is required. It can be split into two statements (an input and an output statement) if the character sequence sent to the terminal differs from the sequence sent from the terminal. Refer to Input/Output Statements - Format Descriptions later in this chapter for more information.

For IBM 3270-compatible terminals, include the following two CURSOR_POS_BEGIN statements:

```
cursor_pos_begin in=11(16)
cursor_pos_begin out=(11(16), 7E(16), 7E(16))
```

The format is:

CURSOR_POS_BEGIN or **CURPB**
INOUT=list of integer, keyword, or **string**
LABEL=string

The INOUT (IO) parameter specifies a character sequence transmitted to and from the terminal. This value is included in the hardware reference manual for your terminal. This parameter is required.

The LABEL (L) parameter indicates if a cursor position transmitted from the terminal requires a response from the application to reposition the cursor, or if the terminal repositions the cursor. If the string is nonblank, cursor positioning requires output from an application. (This is advisable for input devices such as touch panels.) If the string is blank, or you omit the parameter, the terminal positions the cursor.

CURSOR_POS_COLUMN_FIRST

The CURSOR_POS_COLUMN_FIRST statement indicates the column versus row cursor position sequence of your terminal. This statement applies only to terminals for which you specify either BINARY_CURSOR or ANSI_CURSOR on the CURSOR_POS_ENCODING statement. If omitted, it is assumed that the terminal outputs the row first.

The format is:

> **CURSOR_POS_COLUMN_FIRST** or **CURPCF**
> *VALUE=boolean*

The VALUE (V) parameter indicates whether your terminal outputs the column or row first.

Specify TRUE if your terminal has a cursor position sequence that outputs the column before the row.

Specify FALSE if your terminal outputs the row before the column.

If VALUE is omitted, FALSE is assumed.

CURSOR_POS_COLUMN_LENGTH

The CURSOR_POS_COLUMN_LENGTH statement indicates the number of bytes your terminal sends for column values. This statement applies only to terminals for which you specify ANSI_CURSOR on the CURSOR_POS_ENCODING statement. If omitted, it is assumed that the terminal sends a variable number of bytes.

The format is:

> **CURSOR_POS_COLUMN_LENGTH** or **CURPCL**
> *VALUE=integer*

The VALUE (V) parameter indicates the number of bytes your terminal sends for column values.

Enter a number other than 0 only if your terminal is an ANSI terminal and sends a set number of bytes for column values.

If your terminal is not ANSI or if it sends a variable number of bytes, set the value to 0.

If VALUE is omitted, it is assumed that the terminal sends a variable number of bytes.

CURSOR_POS_ENCODING

The CURSOR_POS_ENCODING statement indicates the manner in which your terminal encodes the cursor position. Most terminals use one of the following four types of cursor position encoding.

- ANSI_CURSOR
- BINARY_CURSOR
- CDC721_CURSOR
- IBM3720_CURSOR

These types are described later as values for the TYPE parameter.

If your terminal does not use one of these encoding types, you cannot define the terminal for use with screen mode applications. The CURSOR_POS_ENCODING statement is required.

The format is:

CURSOR_POS_ENCODING or **CURPE**
TYPE=keyword
BIAS=integer

The TYPE (T) specifies the type of encoding used by your terminal. This parameter is required. Which keyword you select for TYPE depends on encoding variables. These variables are used in a sequence that has a general format:

axbyc

Variable	Description
a	The first character string of the cursor position sequence. The value of a is defined in the CURSOR_POS_BEGIN statement.
b	The second character string of the cursor position sequence. The value of b is defined in the CURSOR_POS_SECOND statement.
c	The third character string of the cursor position sequence. The value of c is defined in the CURSOR_POS_THIRD statement.
x	The horizontal position of the cursor.
y	The vertical position of the cursor.

All terminals will have at least an a, x, and y.

Select a keyword value for TYPE from the encoding descriptions that follow:

Keyword	Description
ANSI_CURSOR	Specify this value if your terminal generates the horizontal (x) and vertical (y) cursor positions as decimal graphic characters rather than hexadecimal numbers [12 rather than 0C(16)] in one of the sequences: axby or aybxc
BINARY_CURSOR	Specify this value if your terminal's cursor position sequence includes a bias (described with the BIAS parameter) as follows: a (x+bias) b (y+bias) c or a (y+bias) b (x+bias) c
CDC721_CURSOR	Specify this value if your terminal's cursor position sequence includes a bias (described with the BIAS parameter) and varies depending on the value of the horizontal position of the cursor (x). If x is less than 81, the sequence is: a (x+bias) (y+bias) If x is greater than 80, the sequence is: ab (x+bias-80) (y+bias)
IBM3270_CURSOR	Specify this value for all 3270-compatible terminals.

The BIAS (B) parameter specifies an integer, which is added to the x and y values. The usual number is 32, which is the value of the ASCII space character. The purpose of a bias is to prevent the x and y values from falling in the range of 0 through 31, which has special meaning in communications. The BIAS parameter is required.

Examples:

The Zenith Z19 terminal CURSOR_POS_ENCODING statement is:

```
cursor_pos_encoding bias=(1) type=ansi_cursor
```

The CDC 722 terminal CURSOR_POS_ENCODING statement is:

```
cursor_pos_encoding bias=(32) type=binary_cursor
```

CURSOR_POS_ROW_LENGTH

The CURSOR_POS_ROW_LENGTH statement indicates the number of bytes your terminal sends for row values. This statement applies only to terminals for which you specify ANSI_CURSOR on the CURSOR_POS_ENCODING statement. If omitted, it is assumed that the terminal sends a variable number of bytes.

The format is:

CURSOR_POS_ROW_LENGTH or **CURPRL**
VALUE=integer

The VALUE (V) parameter indicates the number of bytes your terminal sends for row values.

Specify a number other than 0 only if your terminal is an ANSI terminal and sends a set number of bytes for row values.

If your terminal is not ANSI, or if it sends a variable number of bytes, set the value to 0.

If VALUE is omitted, it is assumed that the terminal sends a variable number of bytes.

CURSOR_POS_SECOND

The CURSOR_POS_SECOND statement specifies the second character string of the cursor position sequence. In the general encoding sequence axbyc, this is the variable b (the description of the CURSOR_POS_ENCODING statement provides more information). This statement is required if your terminal uses it.

The format is:

> **CURSOR_POS_SECOND** or **CURPS**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This value is listed in the hardware reference manual for your terminal. This parameter is required.

CURSOR_POS_THIRD

The CURSOR_POS_THIRD statement specifies the third character string of the cursor position sequence. In the general encoding sequence axbyc, this is the variable c (the CURSOR_POS_ENCODING statement provides more information). This statement is required if your terminal uses it.

The format is:

> **CURSOR_POS_THIRD** or **CURPT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This value is listed in the hardware reference manual for your terminal. This parameter is required.

Cursor Behavior Statements

Cursor behavior statements specify how you want the terminal to respond when you move the cursor past the edge of the screen. A brief description of each statement follows. All cursor behavior statements are required. (See the next section for statement formats and detailed descriptions.)

Statement	Description
CHAR_PAST_LAST_ POSITION	Determines cursor movement past the last position on the bottom line of the screen (not using cursor movement keys). This is a required statement.
CHAR_PAST_LEFT/ CHAR_PAST_RIGHT	Determine cursor movement past the left or right edge of the screen (not using cursor movement keys). These are required statements.
MOVE_PAST_BOTTOM/ MOVE_PAST_TOP	Determines cursor movement past the bottom or top edge of the screen by using the cursor movement keys. These are required statements.
MOVE_PAST_LEFT/ MOVE_PAST_RIGHT	Determines cursor movement past the left or right edge of the screen using the cursor movement keys. These are required statements.

Format Descriptions

Each cursor behavior statement has a required TYPE parameter, that determines the cursor movement.

CHAR_PAST_LAST_POSITION

The CHAR_PAST_LAST_POSITION statement determines how the terminal behaves when you move the cursor past the last position on the bottom line of the screen (using keys other than the cursor movement keys). This is a required statement.

The format is:

> **CHAR_PAST_LAST_POSITION** or **CHAPLP**
> **TYPE=keyword**

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword	**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 does not move beyond the bottom edge of the screen.
WRAP_ADJACENT_NEXT	The cursor wraps around to the first column of the top row (home position).
WRAP_SAME_NEXT	The cursor wraps around to the opposite (left) side of the screen and remains on the same line.

CHAR_PAST_LEFT

The CHAR_PAST_LEFT statement determines how the terminal behaves when you move the cursor past the left edge of the screen (using keys other than cursor movement keys). This is a required statement.

The format is:

CHAR_PAST_LEFT or **CHAPL**
  TYPE=keyword

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword Value	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 does not move beyond the left edge of the screen.
WRAP_ADJACENT_NEXT	The cursor reappears at the opposite (right) side on the next line down.
WRAP_SAME_NEXT	The cursor wraps around to the opposite (right) side of the screen and remains on the same line.

CHAR_PAST_RIGHT

The CHAR_PAST_RIGHT statement determines how the terminal behaves when you move the cursor past the right edge of the screen by typing more characters than are allowed on a row. This is a required statement.

The format is:

> **CHAR_PAST_RIGHT** or **CHAPR**
> **TYPE=keyword**

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword	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 does not move beyond the right edge of the screen.
WRAP_ADJACENT_NEXT	The cursor reappears at the opposite (left) side of the screen on the next line down.
WRAP_SAME_NEXT	The cursor wraps around to the opposite (left) side of the screen and remains in the same line.

MOVE_PAST_BOTTOM

The MOVE_PAST_BOTTOM statement determines how the terminal behaves when you move the cursor past the bottom edge of the screen using the cursor movement keys. This is a required statement.

The format is:

MOVE_PAST_BOTTOM or **MOVPB**
  TYPE=keyword

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword	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 does not move beyond the bottom edge of the screen.
WRAP_ADJACENT_NEXT	The cursor wraps around to the top row on the screen and moves one column to the right.
WRAP_SAME_NEXT	The cursor wraps around to the top row on the screen and remains in the same column.

MOVE_PAST_LEFT

The MOVE_PAST_LEFT statement determines how the terminal behaves when you move the cursor past the left edge of the screen by using the cursor movement keys. This is a required statement.

The format is:

MOVE_PAST_LEFT or **MOVPL**
    TYPE=keyword

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword	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 does not move beyond the left edge of the screen.
WRAP_ADJACENT_NEXT	The cursor reappears at the opposite (right) side of the screen on the next line down.
WRAP_SAME_NEXT	The cursor wraps around to the opposite (right) side of the screen and remains on the same line.

MOVE_PAST_RIGHT

The MOVE_PAST_RIGHT statement determines how the terminal behaves when you move the cursor past the right edge of the screen by using the cursor movement keys. This is a required statement.

The format is:

MOVE_PAST_RIGHT or **MOVPR**
  TYPE=keyword

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword Value	**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 does not move beyond the right edge of the screen.
WRAP_ADJACENT_NEXT	The cursor reappears at the opposite (left) side of the screen on the next line down.
WRAP_SAME_NEXT	The cursor wraps around to the opposite (left) side of the screen and remains on the same line.

MOVE_PAST_TOP

The MOVE_PAST_TOP statement determines how the terminal behaves when you move the cursor past the top edge of the screen using the cursor movement keys. This is a required statement.

The format is:

MOVE_PAST_TOP or **MOVPT**
TYPE=keyword

The TYPE (T) parameter determines the movement of the cursor. This parameter is required. The possible values for the cursor position are:

Keyword	**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 does not move beyond the top edge of the screen.
WRAP_ADJACENT_NEXT	The cursor wraps around to the bottom row of the screen and moves one column to the right.
WRAP_SAME_NEXT	The cursor wraps around to the bottom row of the screen and remains in the same column.

Screen Size Specification

The SET_SIZE statement describes the screen size or sizes supported by your terminal and allows you to specify a pick/locate device. Refer to the hardware reference manual for your terminal.

You must specify at least one screen size using the SET_SIZE statement. You can specify up through four screen sizes, one size per SET_SIZE statement.

If your terminal supports more than one screen size, you must set the MULTIPLE_SIZES statement to TRUE; otherwise, set the statement to FALSE.

SET_SIZE

The format is:

> **SET_SIZE** or **SETS**
> **ROWS=integer**
> **COLUMNS=integer**
> *OUT=list of integer, keyword or string*
> *CHARACTER_SPECIFICATION=list of integer*
> *CHARACTER_POSITIONS=list of integer*
> *LINE_SPECIFICATION=list of integer*
> *LINE_POSITIONS=list of integer*
> *DEVICE=string*

The ROWS (R) parameter indicates the number of rows (lines) that your terminal supports. This parameter is required.

The COLUMNS (C) parameter indicates the number of columns (characters) that your terminal supports. This parameter is required.

The OUT (O) parameter specifies a character sequence to be transmitted to the terminal. You obtain this sequence from the hardware reference manual for your terminal. For terminals that can support more than one screen size, this parameter specifies the sequence that is sent to the terminal to switch to the indicated size. Do not specify this parameter if your terminal supports only one screen size.

NOTE

The following five parameters allow you to specify the name and accuracy of a pick/locate device such as a touch panel or mouse. (These parameters are optional.)

The CHARACTER_SPECIFICATION (CS) parameter indicates the starting column, ending column, and character increment for horizontal accuracy.

The CHARACTER_POSITION (CP) parameter indicates the cursor character positions for each pick/locate operation. Use this parameter if the character increment is not consistent between the pick/locate positions.

The LINE_SPECIFICATION (LS) parameter indicates the starting row, ending row, and line increment for vertical accuracy.

The LINE_POSITION (LP) parameter gives the cursor line positions for each pick/locate operation. Use this parameter if the character increment is inconsistent between pick/locate positions.

The DEVICE (D) parameter names the pick/locate device. If omitted, no name is assigned.

Examples

Enter the statements in order of increasing size, giving columns preference over rows. For example, you might enter:

```
set_size rows=24 columns=80 out=(rs dc2 'H' rs dc2 '^')
set_size rows=30 columns=80 out=(rs dc2 'H' rs dc2 '^')
set_size rows=24 columns=132 out=(rs dc2 'G' rs dc2 '^')
set_size rows=30 columns=132 out=(rs dc2 'G' rs dc2 '^')
```

The following example shows how you can specify the SET_SIZE parameters for 80 column mode on a CDC 721 touch panel device. Since this terminal has consistent character increments in 80 column mode, you can use the CHARACTER_SPECIFICATION and LINE_SPECIFICATION parameters. This example specifies a four character increment between columns 11 and 70, and a two line increment between rows 1 and 29.

```
set_size rows=30 columns=80 out=(rs dc2 'H' rs dc2 '^') ..
character_specification=(11,70,4) ..
line_specification=(1,29,2) ..
device='TOUCH_PANEL'
```

The next example shows how you can specify the SET_SIZE parameters for 132 column mode on a CDC 721 touchpanel device. Since this terminal does not have consistent column character increments in 132 column mode (the increment is either 6 or 7 characters), you must use the CHARACTER_POSITIONS parameter. The example specifies each column character increment, and a two line increment between rows 1 and 29.

```
set_size rows=30 columns=132 out=(rs dc2 'G' rs dc2 '^') ..
character_positions=(20,26,33,39,45,51,57,64,70,76,82,88, ..
  95,101,107,113) ..
line_specification=(1,29,2) ..
device='TOUCH_PANEL'
```

Defining Functions and Key Labels for EDIT_FILE

You have three options for defining the programmable function keys for the EDIT_FILE utility:

1. Let EDIT_FILE default to assigning the subcommands and labels associated with the programmable function keys. The defaults used are listed in table 2-1.

2. Use a separate APPLICATION_STRING statement to define each programmable function key. (See the section APPLICATION_STRING Statements for details.)

3. Use the SET_FUNCTION_KEY subcommand in the editor prolog file to define each programmable function key.

Using the APPLICATION_STRING statement is more efficient than using the SET_FUNCTION_KEY subcommand in an editor prolog. However, not all function keys used by EDIT_FILE can be assigned with the APPLICATION_STRING statement. In particular, the shifted and unshifted definitions for the DATA, DOWN, EDIT, FWD, BKW, BACK, HELP, STOP, UNDO, and UP keys must be defined through the SET_FUNCTION_KEY subcommand. If you defined any of these keys for the terminal and want to override the default definition assigned by EDIT_FILE for these keys, follow this procedure:

1. Define the programmable function keys (function keys 1 through 16) through APPLICATION_STRING statements.

2. Create an editor prolog for the definition of these keys by the SET_FUNCTION_KEY subcommand.

Table 2-1. EDIT_FILE Defaults for Function Keys

Cap/Op	Value Used from Terminal Definition
InsCh	INSERT_CHAR with nonblank LABEL, or INSERT_MODE_BEGIN and INSERT_MODE_END with nonblank LABEL
DelCh	DELETE_CHAR with nonblank LABEL
Bkw	BKW with nonblank LABEL, or F1, or F-key with IN the same as BKW IN
First	BKW_S with nonblank LABEL, or F1_S, or F-key with IN the same as BKW_S IN
Fwd	FWD with nonblank LABEL, or F2, or F-key with IN the same as FWD IN
Last	FWD_S with nonblank LABEL, or F2_S, or F-key with IN the same as FWD_S IN
Back	BACK with nonblank LABEL, or F3, or F-key with IN the same as BACK IN
Help	HELP with nonblank LABEL, or F4, or F-key with IN the same as HELP IN
Undo	UNDO with nonblank LABEL, or F5, or F-key with IN the same as UNDO IN
Redo	UNDO_S with nonblank LABEL, or F5_S, or F-key with IN the same as UNDO_S IN (redo is not currently supported by EDIT_FILE)
Quit	STOP with nonblank LABEL, or F6, or F-key with IN the same as STOP IN
Exit	STOP_S with nonblank LABEL, or F6_S, or F-key with IN the same as STOP_S IN
InsLn	INSERT_LINE_BOL or INSERT_LINE_STAY with nonblank LABEL

(Continued)

Table 2-1. EDIT_FILE Defaults for Function Keys *(Continued)*

Cap/Op	Value Used from Terminal Definition
DelLn	DELETE_LINE_BOL or DELETE_LINE_STAY with nonblank LABEL
Home	CURSOR_HOME with nonblank LABEL
OPS	The operations Copy, Move, Mark, Unmrk, MrkCh, MrkBx, Locate, LocNxt, LocAll, Width, Break, Join, and SkpEL cannot be defined through a TDU statement; they are always assigned programmable function keys
ClrEL	ERASE_END_OF_LINE with nonblank LABEL
Middle	This operation cannot be defined through a TDU statement; it is always assigned a programmable function key
Refrsh	ERASE_PAGE_HOME or ERASE_PAGE_STAY with nonblank LABEL
LinUp	UP with nonblank LABEL
LinDn	DOWN with nonblank LABEL
OPS	The operations Format, Center, InsWd, DelWd, InsBk, DelBk, Indent, and Dedent cannot be defined through a TDU statement; they are always assigned programmable function keys

Defining Functions and Key Labels for Applications other than EDIT_FILE

Screen mode applications such as Debug, EDIT_CATALOG, EXPLAIN, IM/Quick, and Programming Environments define both the functions performed and labels assigned to programmable function keys through application menus. You can change the application menu if you want to change either the function key or the label used by these applications. Application menus are described in the NOS/VE Object Code Management manual.

APPLICATION_STRING Statements

These statements are primarily used:

- To define the function of each key in the EDIT_FILE utility. (See Defining Functions and Key Labels for EDIT_FILE.)
- To improve system performance (see the next section, Application Strings for Maximizing System Performance).
- To initialize a terminal (see Initializing Terminals).

The format of the APPLICATION_STRING statement is:

> **APPLICATION_STRING** or **APPS**
>   **NAME=string**
>   **OUT=string**

The NAME (N) parameter specifies the character string that the application associates with the programmable function key. This parameter is required. Values for user-defined applications are listed in the manual that describes the application. Values for the EDIT_ FILE utility follow.

On a statement defining the function of a key in the EDIT_FILE utility, determine the value for the NAME parameter as follows.

- For an unshifted key, enter:

  fse_function_

  followed by the number of the key. For example, the name of the function of unshifted programmable function key F8 is:

  fse_function_8

- For a shifted programmable function key, enter:

  fse_function_shift_

  followed by the number of the key. For example, the name associated with shifted programmable function key F8 is:

  fse_function_shift_8

  On a statement defining the label of a key, the entry is the name of the function of the key (as just described) followed by _LABEL.

- For the unshifted F8 key label, enter:

  FSE_FUNCTION_8_LABEL

- For the shifted F8 key label, enter:

  FSE_FUNCTION_SHIFT_8_LABEL

The OUT (O) parameter specifies the string associated with the value in the NAME parameter. It is sent to the application, which can use it any way it wants. This parameter is required. The OUT parameter can be continued on more than one line under the following conditions:

- Strings that would extend over more than one line must be broken into substrings that the system concatenates. Each substring must be complete on a single line.

- Variables must be complete on each line.

You can use variable names to define lengthy subcommands, as in the following example.

```
f4a='write_file f=$local.t$.$boi,l=m'
f4b='format_cybil_source i=$local.t$.$boi o=$local.t1$.$boi'
f4c='delete_lines l=m'
f4d='read_file f=$local.t1$ p=b'
application_string name=('fse_function_4')..
 out=(f4a ';' f4b ';' f4c ';' f4d )
```

For user-defined applications, refer to the manual that describes the application. Information for the EDIT_FILE utility follows.

When defining the function of a key, the string for the OUT parameter is the subcommand executed when the key is pressed. When you define the label of a key, the string is the label that appears on the screen. Refer to the NOS/VE File Editor manual listed in appendix B for both values.

Application Strings for Maximizing System Performance

There are three application string statements that you can use with any application to maximize the performance of your system.

- The first statement maximizes the speed and efficiency with which your terminal repaints the screen. Without this statement, the terminal repaints screen rows across their entire width when any part of a row needs repainting. If you specify this statement, you use extra CPU resources but the terminal works more efficiently, repainting only those columns that are actually affected.

  The format is:

  ```
  application_string name='optimization' out='true'
  ```

- The second statement is applicable for these terminal definitions:

  CDC_722_30
  DEC_VT100
  DEC_VT100_GOLD
  DEC_VT220
  PC_CONNECT_12
  PC_CONNECT_13
  MAC_CONNECT_11

  It allows you to use the DEC VT100 scrolling regions feature, which makes it possible to scroll vertically through just a portion of screen text. This scrolling regions feature sets up top and bottom margins and issues commands that cause the terminal to scroll up or down within the screen margins.

  To use this feature, specify:

  ```
  application_string name='vt100_scrolling' out='true'
  ```

  This statement is particularly valuable for terminals without insert and delete keys, such as the VT100, because it allows the EDIT_FILE utility to scroll then repaint only one row instead of repainting all rows below the cursor.

- The third statement allows you to use line insertion and deletion commands to scroll the screen. Use this statement with terminals that provide insert and delete capabilities, but lack the VT100 scrolling regions feature described for the preceding statement.

  The format of this statement is:

  ```
  application_string name='insert_delete_scrolling' out='true'
  ```

Initializing Terminals

Most terminals need to be initialized to specify hardware settings for the desired mode of system interaction (screen or line). During initialization, control characters are sent to the terminal through the application statements you specify in your terminal definition to define these settings.

Cursor wraparound is an example of a setting for which your terminal needs to be initialized. In screen mode, you need to suppress cursor wraparound at the edge of the screen for many terminals to prevent unintentional scrolling of the entire screen. In line mode, you need to enable cursor wraparound for many terminals so that you can scroll the entire screen.

Initialization control characters are sent to the terminal to specify the proper settings each time you enter and leave a screen mode application. (For system performance reasons, some users require that control characters be sent to the terminal just once per login; those users should use the INITIALIZE_TERMINAL command which is described in the next section.) The control characters are sent through the following application statements, which you specify in the terminal definition:

- SCREEN_INIT
- SET_SCREEN_MODE
- LINE_INIT
- SET_LINE_MODE

All of these statements are used when you enter and leave each screen mode application.

Each statement lets you define up to 256 characters. You can use additional SCREEN_INIT and LINE_INIT statements if you need to specify more characters. (See the section Screen Mode Application Statements for details on these statements.)

Using the INITIALIZE_TERMINAL Command

For most users, initialization control characters are sent to the terminal every time they enter and exit a screen mode application. Some users have special system performance concerns requiring that initialization control characters be sent to the terminal just once per login. The INITIALIZE_TERMINAL command is designed to handle terminal initialization for these users.

The format of the INITIALIZE_TERMINAL command is:

INITIALIZE_TERMINAL or **INIT**
STATUS=status variable

You can include INITIALIZE_TERMINAL in your user prolog if you choose. Be sure to enter it after you name your terminal model with the CHANGE_TERMINAL_ATTRIBUTES command and select screen or line mode through the CHANGE_INTERACTION_STYLE command. For example:

```
change_terminal_attributes ..
    terminal_model=name of your terminal definition
change_interaction_style style=line or screen
initialize_terminal
```

INITIALIZE_TERMINAL searches the terminal definition for application string statements you set up to initialize the terminal for screen or line mode. It then sends the control characters from these strings to the terminal to change the settings according to the current mode of system interaction.

To initialize the terminal for screen mode, specify control characters through one or more application strings of the following format:

```
application_string name='screen_init' ..
    out='characters sent to the terminal'
```

To initialize the terminal for line mode, specify control characters through one or more application strings of the following format:

```
application_string name='line_init' ..
    out='characters sent to the terminal'
```

Each APPLICATION_STRING statement is limited to 256 characters. If you need to enter more characters, you can use multiple application strings. They will be processed in the order that they appear in your terminal definition. (See the next section for details on the application string statements.)

Screen Mode Application Statements

The statements described in this section apply when you use an application in screen mode; they are ignored for line mode.

A brief description of each statement follows. None of the statements is required. (See the next section for statement formats and detailed descriptions.)

Statement	**Description**
INITIALIZE_TERMINAL	Causes the specified command to be executed each time an application is set to screen mode.
LINE_INIT	Specifies the sequence sent when a terminal user leaves screen mode of an application.
SCREEN_INIT	Specifies the sequence sent when a terminal user enters an application in screen mode.
SET_LINE_MODE	Specifies the string sent when a terminal user leaves screen mode of an application.
SET_SCREEN_MODE	Specifies the string sent when a terminal user enters an application in screen mode.

When you enter an application in screen mode:

- The command specified by an INITIALIZE_TERMINAL statement executes.
- The SET_SCREEN_MODE and SCREEN_INIT statements send character strings to set and clear terminal settings.

When you leave screen mode, the SET_LINE_MODE and LINE_INIT statements send character strings to reset the terminal to the default line mode settings.

The SET_LINE_MODE and LINE_INIT statements are functionally equivalent; however, you can use multiple LINE_INIT statements in a terminal definition but only one SET_LINE_MODE statement. The same is true for the SET_SCREEN_MODE and SCREEN_INIT statements; they are functionally equivalent. You can use multiple SCREEN_INIT statements, but only one SET_SCREEN_MODE statement.

The following example shows the application strings executed during an EDIT_FILE utility session in screen mode.

User Enters:	Statements Executed:
`edit_file file=presto`	• SCREEN_INIT • SET_SCREEN_MODE
`display_value 'hello'`	• SET_LINE_MODE – hello – Press RETURN/NEXT to continue / • SCREEN_INIT • SET_SCREEN_MODE
`deactivate_screen`	• SET_LINE_MODE • LINE_INIT
`activate_screen`	• SCREEN_INIT • SET_SCREEN_MODE
`quit`	• SET_LINE_MODE • LINE_INIT

Format Descriptions

All screen mode application statements (except INITIALIZE_TERMINAL) include a required OUT parameter, which specifies the character sequence for your terminal from the terminal hardware reference manual.

Statements can contain a maximum of 256 characters. If any statement does not fit on one line, you can use continuation lines. If you need to use more than 256 characters in a statement, you can enter as many LINE_INIT and SCREEN_INIT statements in a terminal definition as you need.

INITIALIZE_TERMINAL

The INITIALIZE_TERMINAL statement causes the specified NOS/VE command (for example a CHANGE_TERMINAL_ATTRIBUTES command) to execute automatically when you enter an application in screen mode. The statement can contain a maximum of 256 characters.

The format is:

```
INITIALIZE_TERMINAL or INIT
   SETTA_COMMAND=string
```

The SETTA_COMMAND (SC) parameter specifies the character string containing the NOS/VE command. For example, if you specified the CHANGE_TERMINAL_ATTRIBUTES command, it would automatically set the default terminal attributes.

LINE_INIT

The LINE_INIT statement specifies the sequence sent when a terminal user leaves the screen mode of an application. This statement works the same as SET_LINE_MODE, but it can be specified multiple times in a terminal definition to overcome the 256 character limit on the statement line. If omitted, no special initialization sequence is sent for your terminal.

The format is:

> **LINE_INIT** or **LINI**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

SCREEN_INIT

The SCREEN_INIT statement specifies the sequence sent when a terminal user enters an application in screen mode. This statement works the same as SET_SCREEN_MODE, but it can be specified multiple times in a terminal definition to overcome the 256 character limit on the statement line. If omitted, no special initialization sequence is sent for your terminal.

The format is:

> **SCREEN_INIT** or **SCRI**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

SET_LINE_MODE

The SET_LINE_MODE statement specifies the sequence sent when a terminal user leaves the screen mode of an application.

For example, if you enter the DEACTIVATE_SCREEN subcommand from an EDIT_FILE utility session you move from screen mode to line mode in EDIT_FILE.

If you omit this statement, no special initialization sequence is sent. This statement can appear only once in a terminal definition and can contain a maximum of 256 characters.

The format is:

SET_LINE_MODE or **SETLM**
    OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

SET_SCREEN_MODE

The SET_SCREEN_MODE statement specifies the sequence sent when a terminal user enters an application in screen mode.

For example, if you enter the ACTIVATE_SCREEN subcommand from an EDIT_FILE utility session you move from line mode to screen mode in EDIT_FILE.

If you omit this statement, no special initialization sequence is sent. This statement can appear only once in a terminal definition and can include a maximum of 256 characters.

The format is:

SET_SCREEN_MODE or **SETSM**
    OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

Input/Output Statements

Input/output statements specify character sequences to be sent and/or received by either the terminal or NOS/VE.

A brief description of each statement follows. Required statements are in boldface type. (See the next section for statement formats and detailed descriptions.)

Statement	Description
BACKSPACE	Moves the cursor left one position.
CURSOR_DOWN	Moves the cursor down one line. This statement is required.
CURSOR_HOME	Moves the cursor to the home position.
CURSOR_LEFT	Moves the cursor left one column. This statement is required.
CURSOR_RIGHT	Moves the cursor right one column. This statement is required.
CURSOR_UP	Moves the cursor up one line. This statement is required.
DELETE_CHAR	Deletes the current character and shifts the text remaining on the current line to the left one column.

Statement	Description
DELETE_LINE_BOL	Deletes the current line, shifts the remaining text up, and moves the cursor to the beginning of the line.
DELETE_LINE_STAY	Deletes the current line, shifts the remaining text up, and leaves the cursor where it is.
ERASE_CHAR	Replaces the current character with a space and moves the cursor one column to the left.
ERASE_END_OF_ FIELD	Erases an unprotected field from the cursor position to its end and leaves the cursor where it is.
ERASE_END_OF_LINE	Erases from the cursor position to the end of the line and leaves the cursor where it is.
ERASE_END_OF_ PAGE	Erases everything from the cursor position to the bottom of the screen.
ERASE_FIELD_BOF	Erases the current unprotected field and moves the cursor to the beginning of that unprotected field.

Statement	Description
ERASE_FIELD_STAY	Erases the current unprotected field and leaves the cursor where it is.
ERASE_LINE_BOL	Erases the current line and moves the cursor to the beginning of the blank line.
ERASE_LINE_STAY	Erases the current line and leaves the cursor where it is.
ERASE_PAGE_HOME	Clears the screen and moves the cursor to the home position. This statement is required unless ERASE_PAGE_STAY is used.
ERASE_PAGE_STAY	Clears the screen and leaves the cursor where it is. This statement is required only if ERASE_PAGE_HOME is not used.
ERASE_UNPROTECTED	Erases all the unprotected character positions on the screen.
INSERT_CHAR	Inserts a single blank character at the cursor position and shifts the text remaining on the current line to the right one column.

Statement	Description
INSERT_LINE_BOL	Inserts a blank line before the current line (subsequent lines are moved down) and moves the cursor to the start of the line.
INSERT_LINE_STAY	Inserts a blank line before the current line and leaves the cursor where it is.
INSERT_MODE_BEGIN	Inserts characters the user enters at the cursor position and shifts existing characters to the right.
INSERT_MODE_END	Overwrites existing characters with the characters the user enters.
INSERT_MODE_TOGGLE	Enables switching between insert and overwrite modes.
RESET	Resets the terminal hardware.
TAB_BACKWARD	Moves the cursor to the previous tab stop or unprotected field.
TAB_CLEAR	Clears the tab stop at the cursor position.
TAB_CLEAR_ALL	Clears all tab stops.
TAB_FORWARD	Moves the cursor to the next tab stop or unprotected field.
TAB_SET	Sets a tab stop at the cursor position.

Format Descriptions

All input/output statements, except BACKSPACE, have an INOUT parameter. BACKSPACE has a required IN parameter. The character sequences for these parameters are listed in the hardware reference manual for your terminal.

Use the IN and OUT parameters (rather than INOUT) if you want to specify input and output sequences separately. For example, you could use an IN or OUT parameter alone in a statement if your terminal sends a character sequence different from the one it receives.

A LABEL parameter, which names the keyboard key, is optional for each statement.

Labels on Specific Editing Keys

The information in this subsection applies to the following input/output statements:

CURSOR_HOME

DELETE_CHAR

DELETE_LIN_BOL and DELETE_LINE_STAY (whichever you choose)

ERASE_END_OF_LINE

ERASE_PAGE_HOME and ERASE_PAGE_STAY (whichever you choose)

INSERT_CHAR and INSERT_MODE_BEGIN (whichever you choose)

INSERT_LINE_BOL and INSERT_LINE_STAY (whichever you choose)

If you define the key with an IN or INOUT parameter, the system can respond correctly when the key is pressed. If the LABEL parameter is blank or omitted, the EDIT_FILE application considers the key to be optional and will honor it if it is used. However, EDIT_FILE also offers similar editing operations on a programmable function key. The CDC-supplied definition for the VT100 uses this technique since most VT100s lack these specific keys, although some enhanced VT100s have them.

If the LABEL parameter is present and nonblank, the system assumes that the key is guaranteed to actually exist and does not offer similar editing operations on the programmable function key menu. This allows all available space on the menu to be used for other operations. This technique is used with most CDC-supplied definitions other than the VT100.

BACKSPACE

The BACKSPACE statement specifies the sequence that moves the cursor left one position. This statement is provided for terminals with a backspace key that is different from the CURSOR_LEFT key. If omitted, the terminal does not have this capability.

The format is:

> **BACKSPACE**
> **IN=list of integer, keyword,** or **string**
> *LABEL=string*

The BACKSPACE statement has no abbreviation.

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

CURSOR_DOWN

The CURSOR_DOWN statement specifies the sequence that moves the cursor down one line. This is a required statement.

The format is:

> **CURSOR_DOWN** or **CURD**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

CURSOR_HOME

The CURSOR_HOME statement specifies the sequence that moves the cursor to the home position. This statement is required.

The format is:

> **CURSOR_HOME** or **CURH**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. The output portion of this parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

CURSOR_LEFT

The CURSOR_LEFT statement specifies the sequence that moves the cursor left one column. This is a required statement.

The format is:

> **CURSOR_LEFT** or **CURL**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

CURSOR_RIGHT

The CURSOR_RIGHT statement specifies the sequence that moves the cursor right one column. This is a required statement.

The format is:

> **CURSOR_RIGHT** or **CURR**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

CURSOR_UP

The CURSOR_UP statement specifies the sequence that moves the cursor up one line. This is a required statement.

The format is:

> **CURSOR_UP** or **CURU**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

DELETE_CHAR

The DELETE_CHAR statement specifies the sequence that deletes the current character and shifts the text remaining on the current line to the left one column. If omitted, the terminal does not have this capability.

The format is:

> **DELETE_CHAR** or **DELC**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

DELETE_LINE_BOL

The DELETE_LINE_BOL statement specifies the sequence that deletes the current line, shifts the remaining text up, and moves the cursor to the start of the line. You can use only one of the DELETE_LINE_STAY and DELETE_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **DELETE_LINE_BOL** or **DELLB**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

DELETE_LINE_STAY

The DELETE_LINE_STAY statement specifies the sequence that deletes the current line, shifts the remaining text up, and leaves the cursor where it is. You can use only one of the DELETE_LINE_STAY and DELETE_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **DELETE_LINE_STAY** or **DELLS**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

ERASE_CHAR

The ERASE_CHAR statement specifies the sequence that replaces the current character with a space and moves the cursor one column to the left. If omitted, the terminal does not have this capability.

The format is:

> **ERASE_CHAR** or **ERAC**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_END_OF_FIELD

The ERASE_END_OF_FIELD statement specifies the sequence that erases an unprotected field from the cursor position to its end and leaves the cursor where it is. If omitted, the terminal does not have this capability.

The format is:

> **ERASE_END_OF_FIELD** or **ERAEOF**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_END_OF_LINE

The ERASE_END_OF_LINE statement specifies the sequence that erases from the cursor position to the end of the line and leaves the cursor where it is. If omitted, the terminal does not have this capability.

The format is:

> **ERASE_END_OF_LINE** or **ERAEOL**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

ERASE_END_OF_PAGE

The ERASE_END_OF_PAGE statement specifies the sequence that erases everything from the cursor position to the bottom of the screen. If omitted, the terminal does not have this capability.

The format is:

> **ERASE_END_OF_PAGE** or **ERAEOP**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_FIELD_BOF

The ERASE_FIELD_BOF statement specifies the sequence that erases the current unprotected field and moves the cursor to the beginning of that unprotected field. You can specify only one of the ERASE_FIELD_BOF and ERASE_FIELD_STAY statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **ERASE_FIELD_BOF** or **ERAFB**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_FIELD_STAY

The ERASE_FIELD_STAY statement specifies the sequence that erases the current unprotected field and leaves the cursor where it is. You can use only one of the ERASE_FIELD_BOF and ERASE_FIELD_STAY statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **ERASE_FIELD_STAY** or **ERAFS**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_LINE_BOL

The ERASE_LINE_BOL statement specifies the sequence that erases the current line and moves the cursor to the beginning of the blank line. You can use only one of the ERASE_LINE_STAY and ERASE_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **ERASE_LINE_BOL** or **ERALB**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_LINE_STAY

The ERASE_LINE_STAY statement specifies the sequence that erases the current line and leaves the cursor where it is. You can use only one of the ERASE_LINE_STAY and ERASE_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **ERASE_LINE_STAY** or **ERALS**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

ERASE_PAGE_HOME

The ERASE_PAGE_HOME statement specifies the sequence that clears the screen and moves the cursor to the home position. You can use only one of the ERASE_PAGE_STAY and ERASE_PAGE_HOME statements; one of the two statements is required.

The format is:

> **ERASE_PAGE_HOME** or **ERAPH**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

ERASE_PAGE_STAY

The ERASE_PAGE_STAY statement specifies the sequence that clears the screen and leaves the cursor where it is. You can use only one of the ERASE_PAGE_STAY and ERASE_PAGE_HOME statements; one of the two statements is required.

The format is:

> **ERASE_PAGE_STAY** or **ERAPS**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

ERASE_UNPROTECTED

The ERASE_UNPROTECTED statement specifies the sequence that erases all the unprotected character positions on the screen. If omitted, the terminal does not have this capability.

The format is:

> **ERASE_UNPROTECTED** or **ERAU**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

INSERT_CHAR

The INSERT_CHAR statement specifies the sequence that inserts a single blank character at the cursor position and shifts the text remaining on the current line to the right one column. If omitted, the terminal does not have this capability.

The format is:

> **INSERT_CHAR** or **INSC**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

INSERT_LINE_BOL

The INSERT_LINE_BOL statement specifies the sequence that inserts a blank line before the current line (subsequent lines are moved down) and moves the cursor to the start of the line. You can use only one of the INSERT_LINE_STAY and INSERT_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **INSERT_LINE_BOL** or **INSLB**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

INSERT_LINE_STAY

The INSERT_LINE_STAY statement specifies the sequence that inserts a blank line before the current line (subsequent lines are moved down) and leaves the cursor where it is. You can use only one of the INSERT_LINE_STAY and INSERT_LINE_BOL statements. If you specify neither statement, the terminal does not have this capability.

The format is:

> **INSERT_LINE_STAY** or **INSLS**
>   **INOUT=list of integer, keyword,** or **string**
>   *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

INSERT_MODE_BEGIN

The INSERT_MODE_BEGIN statement specifies the sequence that initiates insert mode, in which characters the users enters are inserted at the cursor position and the existing characters are shifted to the right, rather than being overwritten. If omitted, the terminal does not have this capability.

The format is:

> **INSERT_MODE_BEGIN** or **INSMB**
>   **INOUT=list of integer, keyword,** or **string**
>   *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned; refer to Labels on Specific Editing Keys for further information.

INSERT_MODE_END

The INSERT_MODE_END statement specifies the sequence that ends insert mode. The graphic characters the user enters after this sequence overwrite existing characters. If omitted, the terminal does not have this capability.

The format is:

> **INSERT_MODE_END** or **INSME**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

INSERT_MODE_TOGGLE

The INSERT_MODE_TOGGLE statement specifies the sequence that enables switching between insert and overwrite modes. If omitted, the terminal does not have this capability.

The format is:

> **INSERT_MODE_TOGGLE** or **INSMT**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

RESET

The RESET statement specifies the sequence that resets the terminal hardware. After this sequence is transmitted, the terminal must be reinitialized. If omitted, the terminal does not have this capability.

The format is:

> **RESET** or **RES**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

TAB_BACKWARD

The TAB_BACKWARD statement specifies the sequence that moves the cursor to the previous tab stop or unprotected field. If omitted, the terminal does not have this capability.

The format is:

> **TAB_BACKWARD** or **TABB**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

TAB_CLEAR

The TAB_CLEAR statement specifies the sequence that clears the tab stop at the cursor position. If omitted, the terminal does not have this capability.

The format is:

> **TAB_CLEAR** or **TABC**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

TAB_CLEAR_ALL

The TAB_CLEAR_ALL statement specifies the sequence that clears all tab stops. If omitted, the terminal does not have this capability.

The format is:

> **TAB_CLEAR_ALL** or **TABCA**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

TAB_FORWARD

The TAB_FORWARD statement specifies the sequence that moves the cursor to the next tab stop or unprotected field. If omitted, the terminal does not have this capability.

The format is:

> **TAB_FORWARD** or **TABF**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

TAB_SET

The TAB_SET statement specifies the sequence that sets a tab stop at the cursor position. If omitted, the terminal does not have this capability.

The format is:

> **TAB_SET** or **TABS**
> **INOUT=list of integer, keyword,** or **string**
> *LABEL=string*

The INOUT (IO) parameter specifies a character sequence transmitted to or from NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

Input Statements

Input statements specify character sequences sent by the terminal to NOS/VE.

These statements include:

- CDC standard function key statements
- Programmable function key statements

All input statements have a required IN parameter, with values obtained from the hardware reference manual for your terminal.

CDC Standard Function Key Statements - Definition

CDC standard function keys are keys that, on the CDC 721, have the function written directly on the key. For terminals that lack dedicated standard keys, programmable function keys can be assigned to perform the functions provided by the standard function keys. CDC supports standard function keys in its full screen applications such as EDIT_FILE, EDIT_CATALOG, and ENTER_PROGRAMMING_ENVIRONMENT.

Escape or control sequences, such as ESC H for HELP, can be a good way to define CDC standard function keys. Take care, however, that you do not specify sequences that conflict with terminal hardware sequences, because the program cannot flag this type of error.

CDC standard function key statements are:

Unshifted	Performs Standard Operation	Shifted	Performs Standard Operation
BACK	Yes	BACK_S	No
BKW	Yes	BKW_S	Yes
DATA	No	DATA_S	No
DOWN	No	DOWN_S	No
EDIT	No	EDIT_S	No

Unshifted	Performs Standard Operation	Shifted	Performs Standard Operation
FWD	Yes	FWD_S	Yes
HELP	Yes	HELP_S	No
NEXT	No	NEXT_S	No
STOP	Yes	STOP_S	Yes
UNDO	Yes	UNDO_S	Yes
UP	No	UP_S	No

Control Data supports standardized function keys for its screen mode applications. A standardized function key is one designated to always perform a specific operation. For example, the QUIT key is assigned the same function key in all Control Data-supported screen mode applications.

Standard functions are assigned in the following order:

1. The application must use the standard function.

2. If a dedicated key exists for a particular terminal, the standard function is assigned to that key. Because the key is considered self-explanatory, it is not displayed in a menu.

3. If the dedicated key does not exist, a specific programmable function key is used. The menu of operations displayed at the bottom of each screen shows which programmable function key to press for each function.

The following table lists information about the standard functions. The Operation Label column specifies the name of the operation displayed for each programmable function key on the menu of operations at the bottom of a screen. (Figure 2-1 shows examples of the operation labels on function keys.)

Dedicated Key	Programmable Key	Operation Label	Description of Standard Function
BKW	F1	Bkw	Display previous screen
FWD	F2	Fwd	Display next screen
BKW_S	Shift F1	First	Display first line
FWD_S	Shift F2	Last	Display last line
BACK	F3	Back	Switch to a previously shown display
HELP	F4	Help	Display help

Dedicated Key	Programmable Key	Operation Label	Description of Standard Function
UNDO	F5	Undo	Correct a user error
UNDO_S	Shift F5	Redo	Restore a user operation that was undone
STOP	F6	Quit	Normal termination of the application
STOP_S	Shift F6	Exit	Alternate termination of the application

The function key identifier specifies the key you press to execute the function key. For some terminals, two identifiers (representing shifted and unshifted) are displayed for each menu box. (Figure 2-1 shows examples of function key identifiers.)

Each key identifier is two characters in the following format:

Key Identifier	Description
blanks	Used for shifted function keys that are considered self-explanatory. The LABEL parameter consists of two leading blanks plus at least one nonblank character.
fn	n is the function key number.
kx	x is the numeric keypad symbol. For example, k1 refers to the numeric keypad key 1.
sx	x is the shifted numeric keypad symbol. For example, s2 refers to the shifted key 2 on the numeric keypad.

Key Identifier	Description
px	x is the number of the PF key on the numeric keypad for the VT100 and VT220.
Cn	For PC CONNECT, n is the function key used in conjunction with the CTRL key.
An	For PC CONNECT, n is the key on the numeric/punctuation row of the keyboard used in conjunction with the ALT key.
0x	For DEC_VT100_GOLD terminals, lets you specify more function key combinations in addition to the default key definitions. Press the 0 key first and then the desired keypad key.

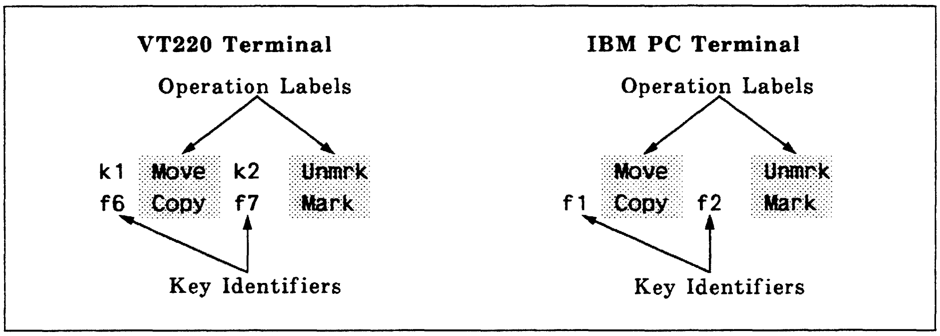


Figure 2-1. Function Key Operation Labels and Key Identifiers

Unshifted	Performs Standard Operation	Shifted	Performs Standard Operation
FWD	Yes	FWD_S	Yes
HELP	Yes	HELP_S	No
NEXT	No	NEXT_S	No
STOP	Yes	STOP_S	Yes
UNDO	Yes	UNDO_S	Yes
UP	No	UP_S	No

CDC supports standardized function keys in its full screen applications. A standardized function key is one that has been designated to always perform a specific operation. For example, the QUIT key is assigned the same function key in all CDC-supported full screen applications.

Standard functions are assigned in the following order:

1. The application must use the standard function.

2. If a dedicated key exists for a particular terminal, the standard function is assigned to that key. Because the key is considered self-explanatory, it is not displayed in a menu.

3. If the dedicated key does not exist, a specific programmable function key is used. Menus displayed on the screen show which programmable function key to use for the standard function

The following table lists information about the standard functions. The Menu Prompt column indicates the menu label used when the programmable function key is displayed.

Dedicated Key	Programmable Key	Menu Prompt	Description of Standard Function
BKW	F1	Bkw	Display previous screen
FWD	F2	Fwd	Display next screen
BKW_S	Shift F1	First	Display first line
FWD_S	Shift F2	Last	Display last line
BACK	F3	Back	Switch to a previously shown display
HELP	F4	Help	Display help

Dedicated Key	Programmable Key	Menu Prompt	Description of Standard Function
UNDO	F5	Undo	Correct a user error
UNDO_S	Shift F5	Redo	Restore a user operation that was undone
STOP	F6	Quit	Normal termination of the application
STOP_S	Shift F6	Exit	Alternate termination of the application

The function key labels indicate the keyboard location of the function key. For some terminals, two labels (nominally representing shifted and unshifted) are displayed for each menu box. Each label is two characters in the following format:

Key Identifier	Description
blanks	Used for shifted function keys that are considered self-explanatory. The LABEL parameter consists of two leading blanks plus at least one nonblank character.
fn	n is the function key number
kx	x is the numeric keypad symbol. For example, k1 refers to the numeric keypad key 1.
sx	x is the shifted numeric keypad symbol. For example, s2 refers to the shifted key 2 on the numeric keypad.
px	x is the number of the PF key on the numeric keypad for the VT100 and VT220.
Cn	For PC Connect, n is the function key used in conjunction with the CTRL key.
An	For PC Connect, n is the key on the numeric/punctuation row of the keyboard used in conjunction with the ALT key.

CDC Standard Function Key Statements - Formats

The CDC standard function key statements are input statements and all require an IN parameter with a value obtained from the hardware reference manual for your terminal.

A LABEL parameter, which names the keyboard key, is optional for each statement.

NOTE

The label can be a maximum of 31 characters. The first two characters are used for key identification in menus for function keys 1 through 16; the remainder of the label should be descriptive and readable. If you do not define a CDC standard key, or if you define it with an IN parameter that matches the IN parameter for one of the programmable function key statements in the CDC-supplied definition, the system assumes that the standard key does not exist as a separate key and substitutes the system-assigned programmable function key.

If you define a CDC standard key with a unique IN parameter and a nonblank LABEL parameter, the system assumes that a separate key exists, and does not assign a programmable function key to perform the operation. The LABEL parameter should be descriptive because some applications may display the label to remind you which keys to press. However, many applications assume that the keyboard is self-explanatory and do not display the LABEL string.

Examples

The following three examples demonstrate the effect of different BACK definitions.

```
back in=(value) label='Back'
```

No programmable function key will be used for the BACK function. Since the LABEL value is nonblank, a dedicated key (which sends the specified value when pressed) is assumed to exist for the BACK function.

```
back in=()
```

Since the LABEL value is blank, a standard programmable function key is used. The standard function key for the BACK operation is F3.

```
bkw in=(bkw-value)

f10 in=(bkw-value) label='10'
```

Since the two IN values are the same, F10 is used instead of the standard F3 for the BACK function.

BACK

The BACK statement specifies the sequence transmitted when you press the BACK key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **BACK** or **BAC**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

BACK_S

The BACK_S statement specifies the sequence transmitted when you press the shifted BACK key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **BACK_S** or **BACS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

BKW

The BKW statement specifies the sequence transmitted when you press the BKW key. If omitted, a full screen application cannot define a function for the key.

The format is:

BKW
IN=list of integer, keyword value, or **string**
LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

BKW_S

The BKW_S statement specifies the sequence transmitted when you press the shifted BKW key. If omitted, a full screen application cannot define a function for the key.

The format is:

BKW_S or **BKWS**
IN=list of integer, keyword value, or **string**
LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

DATA

The DATA statement specifies the sequence transmitted when you press the DATA key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **DATA** or **DAT**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1– through 31–character string that labels the corresponding keyboard key. If omitted, no label is assigned.

DATA_S

The DATA_S statement specifies the sequence transmitted when you press the shifted DATA key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **DATA_S** or **DATS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1– through 31–character string that labels the corresponding keyboard key. If omitted, no label is assigned.

DOWN

The DOWN statement specifies the sequence transmitted when you press the DOWN key. If omitted, a full screen application cannot define a function for the key.

The format is:

DOWN or **DOW**
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

DOWN_S

The DOWN_S statement specifies the sequence transmitted when you press the shifted DOWN key. If omitted, a full screen application cannot define a function for the key.

The format is:

DOWN_S or **DOWS**
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

EDIT

The EDIT statement specifies the sequence transmitted when you press the EDIT key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **EDIT** or **EDI**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

EDIT_S

The EDIT_S statement specifies the sequence transmitted when you press the shifted EDIT key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **EDIT_S** or **EDIS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

FWD

The FWD statement specifies the sequence transmitted when you press the FWD key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **FWD**
>   **IN=list of integer, keyword value,** or **string**
>   *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

FWD_S

The FWD_S statement specifies the sequence transmitted when you press the shifted FWD key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **FWD_S** or **FWDS**
>   **IN=list of integer, keyword value,** or **string**
>   *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

HELP

The HELP statement specifies the sequence transmitted when you press the HELP key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **HELP** or **HEL**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1– through 31–character string that labels the corresponding keyboard key. If omitted, no label is assigned.

HELP_S

The HELP_S statement specifies the sequence transmitted when you press the shifted HELP key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **HELP_S** or **HELS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1– through 31–character string that labels the corresponding keyboard key. If omitted, no label is assigned.

NEXT

The NEXT statement specifies the sequence transmitted when you press the NEXT key. If omitted, a screen mode application cannot define a function for the key. This statement is useful only through the LABEL parameter; otherwise, some applications may assign one of the programmable function keys with the Next label.

This statement does not specify the character used to transmit text, since CDCNET uses the Transparent Forwarding Character connection attribute to determine that character.

The format is:

> **NEXT** or **NEX**
> **IN=13** (ASCII carriage return)
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required, but ignored.

If you have an asynchronous terminal that uses some character other than the ASCII carriage return code, add these commands to your user prolog:

```
change_terminal_attributes end_line_character=value
change_connection_attributes terminal_file_name=input ..
    transparent_forward_character=value
change_term_conn_defaults tfc=value
```

You do not need to use these commands if you have an IBM 3270 and are using CDCNET.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

NEXT_S

The NEXT_S statement specifies the sequence transmitted when you press the shifted NEXT key. If omitted, a screen mode application cannot define a function for the key. This statement is not required for any CDC-supplied applications, but user-written applications might make use of it.

The format is:

> **NEXT_S** or **NEXS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1– through 31–character string that labels the corresponding keyboard key. If omitted, no label is assigned.

STOP

The STOP statement specifies the sequence transmitted when you press the STOP key. Sequences for terminating operations are specified in the NOS/VE System Usage manual. If omitted, a screen mode application cannot define a function for the key.

The format is:

> **STOP** or **STO**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

STOP_S

The STOP_S statement specifies the sequence transmitted when you press the shifted STOP key. Sequences for terminating operations are specified in the NOS/VE System Usage manual. If omitted, a screen mode application cannot define a function for the key.

The format is:

> **STOP_S** or **STOS**
> **IN=list of integer, keyword value,** or **string**
> *LABEL=string*

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

UNDO

The UNDO statement specifies the sequence transmitted when you press the UNDO key. If omitted, a full screen application cannot define a function for the key.

The format is:

UNDO or **UND**
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

UNDO_S

The UNDO_S statement specifies the sequence transmitted when you press the shifted UNDO key. If omitted, a full screen application cannot define a function for the key.

The format is:

UNDO_S or **UNDS**
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

UP

The UP statement specifies the sequence transmitted when you press the UP key. If omitted, a full screen application cannot define a function for the key.

The format is:

UP
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

UP_S

The UP_S statement specifies the sequence transmitted when you press the shifted UP key. If omitted, a full screen application cannot define a function for the key.

The format is:

UP_S or **UPS**
  IN=list of integer, keyword value, or **string**
  LABEL=string

The IN (I) parameter specifies a character sequence transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that labels the corresponding keyboard key. If omitted, no label is assigned.

Programmable Function Key Statements - Definition

All system-defined screen mode applications use programmable function keys that tell the application what to do next. For example, programmable function keys in the EDIT_FILE utility allow you to execute frequently used subcommands by pressing a function key (or a sequence of keys specific to the terminal).

You define what input sequence the terminal sends upline to be recognized as a programmable function key. You must define 16 programmable function keys (any combination of shifted and unshifted keys). You should define all possible key presses for your terminal.

If your terminal is not an IBM 3270-compatible device and does not have programmable function keys, you can define function keys using a control sequence, such as the ESCAPE (ESC) key, in combination with the number keys.

For example:

Unshifted	Shifted
ESC 1	ESC shifted 1
ESC 2	ESC shifted 2
⋮	⋮
ESC 9	ESC shifted 9

This definition scheme gives you a maximum of 18 programmable function keys. Be sure not to use any sequences that conflict with terminal hardware sequences.

If local screen formatting applications use programmable function keys to drive menus, these keys must be defined in the terminal definition for your terminal.

Special Considerations for the IBM 3270 Function Keys

The design of the IBM 3270 terminal hardware does not allow you to use the control sequence definition scheme described previously. You can use only the 12 or 24 function keys that are physically on the terminal. The released terminal definitions for the 3270 terminals assume there are 24 function keys. If your terminal has only 12 function keys, they will be defined as keys F1 through F12; the shifted function keys (F13 through F24) will not be available. Some software applications may assign vital operations to shifted function keys. Look at the applications you will be using to determine whether you need to redefine the 12 function keys.

Programmable Function Key Statements - Formats

There are two format types for programmable function key statements:

- Fn statements for unshifted function keys F1 through F16.
- Fn_S for shifted function keys F1 through F16.

These statements are input statements that require an IN parameter including the sequence listed in the hardware reference manual for your terminal.

A LABEL parameter, which names the keyboard key, is optional for each statement.

NOTE

The LABEL parameter can be a maximum of 31 characters telling the system that the key is available. Most applications use the first two characters of the LABEL parameter as a key identifier in the function key menu displayed at the bottom of a screen to help the terminal operator recognize which key to press (see the description of key identifiers under CDC Standard Function Key Statements earlier in this chapter). If you do not want a LABEL parameter to be displayed, use two leading blanks followed by at least one nonblank character.

Fn Statements

The Fn statements specify the sequence transmitted when you press an unshifted Fn programmable function key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **Fn**
> **IN=list of integer, keyword,** or **string**
> *LABEL=string*

n can be from 1 through 16.

The IN (I) parameter specifies a character sequence to be transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that names the corresponding keyboard key. Most screen applications use the presence or absence of a nonblank label to determine which keys can be assigned functions.

Fn_S Statements

The Fn_S statements specify the character sequence to be transmitted when you press a shifted Fn programmable function key. If omitted, a full screen application cannot define a function for the key.

The format is:

> **Fn_S** or **FnS**
> **IN=list of integer, keyword,** or **string**
> *LABEL=string*

n can be from 1 through 16.

The IN (I) parameter specifies a character sequence to be transmitted to NOS/VE. This parameter is required.

The LABEL (L) parameter specifies a 1- through 31-character string that names the corresponding keyboard key. Most screen applications use the presence or absence of a nonblank label to determine which keys can be assigned functions.

Output Statements

Output statements specify character sequences to be sent from NOS/VE to the terminal. They are divided into four types: send statements, physical terminal attribute statements, logical terminal attribute statements, and line drawing statements.

- Send statements specify sequences sent to the terminal.
- Physical terminal attribute statements define the physical attributes of the terminal, such as blinking and inverse video.
- Logical terminal attribute statements define various types of fields on the screen.
- Line drawing statements specify line weights and characters for drawing lines.

Send Statements - Overview

The send statements specify sequences sent to the terminal by NOS/VE.

A brief description of each statement follows. None of these statements is required. (See the next section for statement formats and detailed descriptions.)

Statement	Description
BELL_ACK	Rings the alternate bell.
BELL_NAK	Rings the bell on an error.
DISPLAY_BEGIN	Enables the display (characters received show on the screen).
DISPLAY_END	Disables the display (characters received do not show on the screen).
OUTPUT_BEGIN	Allows output to begin. NOS/VE sends this sequence before starting output (after receiving input).
OUTPUT_END	Ends output. NOS/VE sends this sequence after ending output (before receiving input).
PRINT_BEGIN	Enables the printer (characters received are printed).
PRINT_END	Disables the printer (characters received are not printed).
PROTECT_ALL	Protects all character positions on the screen.
RETURN	Moves the cursor to the beginning of the current line.

Send Statements - Format Descriptions

The send statements are output statements. Each has a required OUT parameter that specifies a character string you obtain from the hardware reference manual for your terminal.

BELL_ACK

The BELL_ACK statement specifies the sequence that rings the alternate bell. If omitted, the alternate bell is disabled.

The format is:

BELL_ACK or **BELA**
OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies specifies a character sequence transmitted to the terminal. This parameter is required.

BELL_NAK

The BELL_NAK statement specifies the sequence that rings the bell on an error. If omitted, the ASCII BEL character (7) is transmitted.

The format is:

BELL_NAK or **BELN**
OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies specifies a character sequence transmitted to the terminal. This parameter is required.

DISPLAY_BEGIN

The DISPLAY_BEGIN statement specifies the sequence that enables the display (characters received show on the screen). If omitted, no sequence needs to be sent for the terminal to display characters.

NOTE

DISPLAY_BEGIN is not used by NOS/VE, but is accepted for compatibility with terminal definitions converted from NOS.

The format is:

> **DISPLAY_BEGIN** or **DISB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

DISPLAY_END

The DISPLAY_END statement specifies the sequence that disables the display (characters received do not show on the screen). If omitted, no sequence needs to be sent to disable displaying characters.

NOTE

DISPLAY_END is not used by NOS/VE, but is accepted for compatibility with terminal definitions converted from NOS.

The format is:

> **DISPLAY_END** or **DISE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

OUTPUT_BEGIN

The OUTPUT_BEGIN statement specifies the sequence that allows output to begin. NOS/VE sends the specified sequence before starting output (after receiving input). The statement should include the sequence to disable protected areas if it is supported by the terminal. If omitted, no sequence needs to be sent to begin output.

The format is:

> **OUTPUT_BEGIN** or **OUTB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

OUTPUT_END

The OUTPUT_END statement specifies the sequence that ends output. NOS/VE sends the sequence specified after ending output (before receiving input). It should include the sequence to enable protected areas if the terminal supports protected areas. If omitted, no sequence needs to be sent to end output.

The format is:

> **OUTPUT_END** or **OUTE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

PRINT_BEGIN

The PRINT_BEGIN statement specifies the sequence that enables the printer (it prints received characters). If omitted, no sequence needs to be sent to print characters.

The format is:

> **PRINT_BEGIN** or **PRIB**
> **OUT=list of integer, keyword,** or .**string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

PRINT_END

The PRINT_END statement specifies the sequence that disables the printer (it does not print received characters). If omitted, no sequence needs to be sent to disable the printer.

The format is:

> **PRINT_END** or **PRIE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

PROTECT_ALL

The PROTECT_ALL statement specifies the sequence that protects character positions on the screen. If omitted, the terminal does not have this capability.

The format is:

> **PROTECT_ALL** or **PROA**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

RETURN

The RETURN statement specifies the sequence that moves the cursor to the beginning of the line it is at. If omitted, the terminal does not have this capability.

The format is:

> **RETURN** or **RET**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

Physical Terminal Attribute Statements - Overview

The physical terminal attribute statements define physical attributes of the terminal. Some terminals, however, use a character position on the screen to enable/disable these attributes. If this is the case with your terminal, do not use these statements.

A brief description of each statement follows. None of the statements are required. (See the next section for statement formats and detailed descriptions.)

Statement	Description
ALT_BEGIN	Displays characters in the alternate intensity (bright or dim).
ALT_END	Stops the display of characters in the alternate intensity (bright or dim).
BLINK_BEGIN	Blinks characters.
BLINK_END	Stops the blinking of characters.
HIDDEN_BEGIN	Stops the display of characters (sets up hidden fields, as for passwords).
HIDDEN_END	Starts the display of characters.
HIGH_ INTENSITY_ BEGIN	Indicates the character sequence that begins the high intensity highlighting style.
HIGH_ INTENSITY_END	Indicates the character sequence which ends the high intensity highlighting style.
INVERSE_BEGIN	Displays characters in inverse video.

Statement	Description
INVERSE_END	Stops the display of characters in inverse video.
LOW_ INTENSITY_ BEGIN	Indicates the character sequence that begins the low intensity highlighting style.
LOW_ INTENSITY_END	Indicates the character sequence that ends the low intensity highlighting style.
PROTECT_BEGIN	Protects character positions written to.
PROTECT_END	Stops the protection of character positions written to.
UNDERLINE_ BEGIN	Underlines characters.
UNDERLINE_END	Stops the underlining of characters.

Physical Terminal Attribute Statements - Format Descriptions

The physical terminal attribute statements are output statements. Each has a required OUT parameter that specifies a character string you obtain from the hardware reference manual for your terminal.

ALT_BEGIN

The ALT_BEGIN statement specifies the sequence that displays characters in the alternate intensity (either bright or dim). If omitted, no sequence can be sent to display characters in an alternate intensity.

The format is:

ALT_BEGIN or **ALTB**
  OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

ALT_END

The ALT_END statement specifies the sequence that stops the display of characters in the alternate intensity (either bright or dim). If omitted, no sequence can be sent to stop displaying characters in the alternate intensity.

The format is:

ALT_END or **ALTE**
  OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

BLINK_BEGIN

The BLINK_BEGIN statement specifies the sequence that blinks characters. If omitted, no sequence can be sent to blink characters.

The format is:

> **BLINK_BEGIN** or **BLIB**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

BLINK_END

The BLINK_END statement specifies the sequence that stops the blinking of characters. If omitted, no sequence can be sent to stop the blinking of characters.

The format is:

> **BLINK_END** or **BLIE**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

HIDDEN_BEGIN

The HIDDEN_BEGIN statement specifies the sequence that stops the display of characters (sets up hidden fields, as for passwords). The HAS_HIDDEN statement must also be specified to set up hidden fields. If HIDDEN_BEGIN is omitted, no sequence can be sent to start hidden fields.

The format is:

HIDDEN_BEGIN or **HIDB**
  OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

HIDDEN_END

The HIDDEN_END statement specifies the sequence that starts the display of characters. The HAS_HIDDEN statement must also be specified to set up hidden fields. If HIDDEN_END is omitted, no sequence can be sent to end hidden fields.

The format is:

HIDDEN_END or **HIDE**
  OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

HIGH_INTENSITY_BEGIN

The HIGH_INTENSITY_BEGIN statement indicates the character sequence that begins the high intensity highlighting on the terminal screen.

The format is:

HIGH_INTENSITY_BEGIN or **HIGIB**
OUT=list of integer, keyword

The OUT (O) parameter specfies a character sequence transmitted to the terminal to begin high intensity.

HIGH_INTENSITY_END

The HIGH_INTENSITY_END statement indicates the character sequence that ends the high intensity highlighting on the terminal screen.

The format is:

HIGH_INTENSITY_END or **HIGIE**
OUT=list of integer, keyword

The OUT (O) parameter specfies a character sequence transmitted to the terminal to end high intensity.

INVERSE_BEGIN

The INVERSE_BEGIN statement specifies the sequence that displays characters in inverse video. If omitted, no sequence can be sent to start inverse video. The last example in the following section on Logical Terminal Attributes explains how to produce marking in inverse video in the EDIT_FILE utility.

The format is:

> **INVERSE_BEGIN** or **INVB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

INVERSE_END

The INVERSE_END statement specifies the sequence that stops the display of characters in inverse video. If omitted, no sequence can be sent to end inverse video.

The format is:

> **INVERSE_END** or **INVE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

LOW_INTENSITY_BEGIN

The LOW_INTENSITY_BEGIN statement indicates the character sequence that begins the low intensity highlighting on the terminal screen.

The format is:

> **LOW_INTENSITY_BEGIN** or **LOWIB**
> **OUT=list of integer, keyword**

The OUT (O) parameter specifies a character sequence transmitted to the terminal to begin low intensity.

LOW_INTENSITY_END

The LOW_INTENSITY_END statement indicates the character sequence that ends the low intensity highlighting on the terminal screen.

The format is:

> **LOW_INTENSITY_END** or **LOWIE**
> **OUT=list of integer, keyword**

The OUT (O) parameter specifies a character sequence transmitted to the terminal to end low intensity.

PROTECT_BEGIN

The PROTECT_BEGIN statement specifies the sequence that protects character positions written to. To set up protected fields, you must also specify the HAS_PROTECT statement. If PROTECT_BEGIN is omitted, no sequence can be sent to start a protected field.

The format is:

> **PROTECT_BEGIN** or **PROB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

PROTECT_END

The PROTECT_END statement specifies the sequence that ends protection of character positions written to. If omitted, no sequence can be sent to end a protected field.

The format is:

> **PROTECT_END** or **PROE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

UNDERLINE_BEGIN

The UNDERLINE_BEGIN statement specifies the sequence that underlines characters. If omitted, no sequence can be sent to start underlining.

The format is:

> **UNDERLINE_BEGIN** or **UNDB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

UNDERLINE_END

The UNDERLINE_END statement specifies the sequence that stops underlining of characters. If omitted, no sequence can be sent to stop underlining.

The format is:

> **UNDERLINE_END** or **UNDE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies a character sequence transmitted to the terminal. This parameter is required.

Logical Terminal Attribute Statements - Overview

Logical attributes are used mainly by full screen applications to define various types of fields on the screen. You can define a logical attribute as a physical attribute or as a combination of physical attributes.

The 12 logical terminal attribute statements are:

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

None of these statements is required.

The logical terminal attribute statements allow you to uniquely identify errors, input, italics, messages, output, and titles for each terminal. All applications using terminal definitions for these attributes will (when the user uses the same terminal) look alike in these areas. Usability is improved when the user can transfer this type of knowledge from one application to another. If you do not specify sequences for these attributes and the application does not define its own, text output prints normally.

Examples:

- A full screen application defines all input parameters as the logical type INPUT_TEXT. Then in the terminal definition, you specify that the INPUT_TEXT_BEGIN statement has the physical characteristic of underlining. When an application uses the sequence assigned to INPUT_TEXT_BEGIN for input fields, it gets underlined input fields. (The file also contains the INPUT_ TEXT_END statement to define the stopping of underlining.)

- The applications available on NOS/VE do not always use the logical attributes you set up. Some applications have special needs, for which they define logical attributes that override those you defined. This happens with the use of INPUT_TEXT in the EDIT_FILE utility. If you set up all input to be underlined in the terminal definition, in the editor your screen would often be filled with underlines. Since this is not acceptable, the editor overrides any sequences specified for INPUT_TEXT_BEGIN and INPUT_TEXT_END.

- Most terminals do not have italics, but you can assign physical characteristics to the italic statements, so that when an application uses italics the terminal will respond. The EDIT_FILE utility uses this capability to produce marking in inverse video. If you want to use marking in the editor, you should include the ITALIC_BEGIN and ITALIC_END statements in the terminal definition. On the OUT parameter of these statements, specify the character sequences that start and stop a clearly visible video attribute (such as inverse video) for your terminal.

  For the CDC 721 terminal, the following statements are included.

  ```
  italic_begin out=(start_inverse)
  italic_end out=(stop_inverse)
  ```

  The variable name START_INVERSE is defined at the beginning of the definition as `rs 'D'` and the variable name STOP_INVERSE is defined as `rs 'E'`.

Logical Terminal Attribute Statements - Format Descriptions

The logical terminal attribute statements are output statements. Each has a required OUT parameter that specifies a character sequence that represents physical characteristics. You obtain this sequence from the hardware reference manual for your terminal.

ERROR_BEGIN

The ERROR_BEGIN statement specifies the sequence sent to begin an error field. If omitted, any text output in this field prints normally.

The format is:

> **ERROR_BEGIN** or **ERRB**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

ERROR_END

The ERROR_END statement specifies the sequence sent to end an error field. If omitted, text output continues as specified by the ERROR_BEGIN statement.

The format is:

> **ERROR_END** or **ERRE**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

INPUT_TEXT_BEGIN

The INPUT_TEXT_BEGIN statement specifies the sequence sent to begin an input field. Your terminal may support protected fields by using a video attribute (such as alternate intensity) for unprotected areas of the screen. If it does, you should define INPUT_TEXT_BEGIN and INPUT_TEXT_END so that screen formatting applications display the input fields correctly as unprotected areas. If omitted, any text output in this field prints normally.

The format is:

> **INPUT_TEXT_BEGIN** or **INPTB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

INPUT_TEXT_END

The INPUT_TEXT_END statement specifies the sequence sent to end an input field. If omitted, text output continues as specified by the INPUT_TEXT_BEGIN statement.

The format is:

> **INPUT_TEXT_END** or **INPTE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

ITALIC_BEGIN

The ITALIC_BEGIN statement specifies the sequence sent to begin italics. If your terminal supports an alternate character set, here is where you can make use of it with screen formatting applications. To use marking in the EDIT_FILE utility, italics must be set to some clearly visible video attribute, such as inverse video. If omitted, any text output in this field prints normally.

The format is:

> **ITALIC_BEGIN** or **ITAB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

ITALIC_END

The ITALIC_END statement specifies what sequence is transmitted to end italics. If omitted, text output continues as specified by the ITALIC_BEGIN statement.

The format is:

> **ITALIC_END** or **ITAE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

MESSAGE_BEGIN

The MESSAGE_BEGIN statement specifies what sequence is transmitted to begin a message field. The display attributes specified here are used when printing help and similar information. If omitted, any text output in this field prints normally.

The format is:

> **MESSAGE_BEGIN** or **MESB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

MESSAGE_END

The MESSAGE_END statement specifies what sequence is transmitted to end a message field. If omitted, text output continues as specified by the MESSAGE_BEGIN statement.

The format is:

> **MESSAGE_END** or **MESE**
> **OUT=list of integer** or **keyword**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

OUTPUT_TEXT_BEGIN

The OUTPUT_TEXT_BEGIN statement specifies what sequence is transmitted to begin an output field. If omitted, any text output in this field prints normally.

The format is:

> **OUTPUT_TEXT_BEGIN** or **OUTTB**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

OUTPUT_TEXT_END

The OUTPUT_TEXT_END statement specifies what sequence is transmitted to end an output field. If omitted, text output continues as specified by the OUTPUT_TEXT_BEGIN statement.

The format is:

> **OUTPUT_TEXT_END** or **OUTTE**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

TITLE_BEGIN

The TITLE_BEGIN statement specifies what sequence is transmitted to begin a title field. If omitted, any text output in this field prints normally.

The format is:

> **TITLE_BEGIN** or **TITB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

TITLE_END

The TITLE_END statement specifies the sequence sent to end a title field. If omitted, text output continues as specified by the TITLE_BEGIN statement.

The format is:

> **TITLE_END** or **TITE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence transmitted to the terminal. This parameter is required.

Line Drawing Statements - Overview

Screen formatting applications allow you to specify:

- Three weights of line drawing (fine, medium, and bold).
- Output sequences for each weight (on and off).
- Characters for horizontal lines, vertical lines, box corners, and box intersections.

The following statements can be used to specify line drawings for the three line weights. Different statements specify the beginning and end of a line weight, horizontal and vertical lines, the four box corners, and intersection characters.

Weight	Line Drawing Statement
Fine	LD_FINE_BEGIN
	LD_FINE_CROSS
	LD_FINE_DOWN_T
	LD_FINE_END
	LD_FINE_HORIZONTAL
	LD_FINE_LEFT_T
	LD_FINE_LOWER_LEFT
	LD_FINE_LOWER_RIGHT
	LD_FINE_RIGHT_T
	LD_FINE_UP_T
	LD_FINE_UPPER_LEFT
	LD_FINE_UPPER_RIGHT
	LD_FINE_VERTICAL

Weight	Line Drawing Statement
Medium	LD_MEDIUM_BEGIN
	LD_MEDIUM_CROSS
	LD_MEDIUM_DOWN_T
	LD_MEDIUM_END
	LD_MEDIUM_HORIZONTAL
	LD_MEDIUM_LEFT_T
	LD_MEDIUM_LOWER_LEFT
	LD_MEDIUM_LOWER_RIGHT
	LD_MEDIUM_RIGHT_T
	LD_MEDIUM_UP_T
	LD_MEDIUM_UPPER_LEFT
	LD_MEDIUM_UPPER_RIGHT
	LD_MEDIUM_VERTICAL
Bold	LD_BOLD_BEGIN
	LD_BOLD_CROSS
	LD_BOLD_DOWN_T
	LD_BOLD_END
	LD_BOLD_HORIZONTAL
	LD_BOLD_LEFT_T
	LD_BOLD_LOWER_LEFT
	LD_BOLD_LOWER_RIGHT
	LD_BOLD_RIGHT_T
	LD_BOLD_UP_T
	LD_BOLD_UPPER_LEFT
	LD_BOLD_UPPER_RIGHT
	LD_BOLD_VERTICAL

Line Drawing Statements - Format Descriptions

If you can actually draw lines on your terminal (rather than improvising lines using such characters as the hyphen), place the sequences to turn the line drawing on and off in the BEGIN and END statements. You can specify the same sequences for all three line weights, if your terminal has only one line weight. If your terminal has only two line weights, you can specify the same sequences for two of the sets of statements.

If your terminal has no bold lines, you can improvise a bold line drawing character set. Define all characters as blanks (' ') and use the sequences defined in the INVERSE_BEGIN and INVERSE_END statements as the LD_BOLD_BEGIN and LD_BOLD_END character sequences (INVERSE_BEGIN and INVERSE_END are physical terminal attribute statements).

If you cannot actually draw lines on your terminal, use:

- The hyphen character for a horizontal line.
- The colon or a similar character for a vertical line.
- Either the asterisk or plus character for corners and intersections.

Since the BEGIN and END statements would be blank, you could equate them to a terminal attribute such as blinking (use the character sequence set up in the BLINK_BEGIN and BLINK_END statements).

The line drawing statements are output statements. Each has an OUT parameter that specifies a character sequence listed in the hardware reference manual for your terminal.

LD_FINE_BEGIN

The LD_FINE_BEGIN statement specifies the sequence sent to start a fine line. If omitted, no sequence needs to be sent to start a fine line.

The format is:

LD_FINE_BEGIN or **LDFB**
OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_CROSS

The LD_FINE_CROSS statement defines the character drawn for fine lines at the point where the lines cross. If omitted, the terminal does not have this capability.

The format is:

LD_FINE_CROSS or **LDFC**
OUT=list of integer, keyword, or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_DOWN_T

The LD_FINE_DOWN_T statement specifies the sequence that defines the character drawn for fine lines at the meeting point of the horizontal line and a line that originates at the horizontal line and goes upward. The following character appears on the screen:

⊥

If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_DOWN_T** or **LDFDT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_END

The LD_FINE_END statement specifies the sequence sent to end a fine line. If omitted, no sequence needs to be sent to end a fine line.

The format is:

> **LD_FINE_END** or **LDFE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_HORIZONTAL

The LD_FINE_HORIZONTAL statement defines the character for drawing fine horizontal lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_HORIZONTAL** or **LDFH**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_LEFT_T

The LD_FINE_LEFT_T statement specifies the sequence that defines the character drawn for fine lines at the meeting point of the vertical line and a line that originates at the vertical line and goes to the right. The following character appears on the screen:

├

If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_LEFT_T** or **LDFLT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_LOWER_LEFT

The LD_FINE_LOWER_LEFT statement defines the character drawn for fine lines at the lower left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_LOWER_LEFT** or **LDFLL**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_LOWER_RIGHT

The LD_FINE_LOWER_RIGHT statement defines the character drawn for fine lines at the lower right corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_LOWER_RIGHT** or **LDFLR**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_RIGHT_T

The LD_FINE_RIGHT_T statement specifies the sequence that defines the character drawn for fine lines at the meeting point of the vertical line and a line that originates at the vertical line and goes to the left. The following character appears on the screen:

┤

If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_RIGHT_T** or **LDFRT**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_UP_T

The LD_FINE_UP_T statement specifies the sequence that defines the character drawn for fine lines at the meeting point of the horizontal line and a line that originates at the horizontal line and goes down. The following character appears on the screen:

┬

If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_UP_T** or **LDFUT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_UPPER_LEFT

The LD_FINE_UPPER_LEFT statement defines the character drawn for fine lines at the upper left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_UPPER_LEFT** or **LDFUL**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_UPPER_RIGHT

The LD_FINE_UPPER_RIGHT statement defines the character drawn for fine lines at the upper right corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_UPPER_RIGHT** or **LDFUR**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_FINE_VERTICAL

The LD_FINE_VERTICAL statement defines the character drawn for fine vertical lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_FINE_VERTICAL** or **LDFV**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_BEGIN

The LD_MEDIUM_BEGIN statement specifies the sequence sent to start a medium line. If omitted, no sequence needs to be sent to start a medium line.

The format is:

> **LD_MEDIUM_BEGIN** or **LDMB**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_CROSS

The LD_MEDIUM_CROSS statement defines the character drawn for medium lines at the point where the lines cross. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_CROSS** or **LDMC**
>   **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_DOWN_T

The LD_MEDIUM_DOWN_T statement specifies the sequence that defines the character drawn for medium lines at the meeting point of a horizontal line and a line that originates at the horizontal line and goes upward. The following character appears on the screen:

⊥

If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_DOWN_T** or **LDMDT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_END

The LD_MEDIUM_END statement specifies the sequence sent to end a medium line. If omitted, no sequence needs to be sent to end a medium line.

The format is:

> **LD_MEDIUM_END** or **LDME**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_HORIZONTAL

The LD_MEDIUM_HORIZONTAL statement defines the character drawn for medium horizontal lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_HORIZONTAL** or **LDMH**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_LEFT_T

The LD_MEDIUM_LEFT_T statement specifies the sequence that defines the character drawn for medium lines at the meeting point of a vertical line and a line that originates at the vertical line and goes to the right. The following character appears on the screen:

├

If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_LEFT_T** or **LDMLT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_LOWER_LEFT

The LD_MEDIUM_LOWER_LEFT statement defines the character drawn for medium lines at the lower left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_LOWER_LEFT** or **LDMLL**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_LOWER_RIGHT

The LD_MEDIUM_LOWER_RIGHT statement defines the character drawn for medium lines at the lower right corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_LOWER_RIGHT** or **LDMLR**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_RIGHT_T

The LD_MEDIUM_RIGHT_T statement specifies the sequence that defines the character drawn for medium lines at the meeting point of the vertical line and a line that originates at the vertical line and goes to the left. The following character appears on the screen:

┤

If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_RIGHT_T** or **LDMRT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_UP_T

The LD_MEDIUM_UP_T statement specifies the sequence that defines the character drawn for medium lines at the meeting point of the horizontal line and a line that originates at the horizontal line and goes downward. The following character appears on the screen:

┬

If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_UP_T** or **LDMUT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_UPPER_LEFT

The LD_MEDIUM_UPPER_LEFT statement defines the character drawn for medium lines at the upper left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is as follows:

> **LD_MEDIUM_UPPER_LEFT** or **LDMUL**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_UPPER_RIGHT

The LD_MEDIUM_UPPER_RIGHT statement defines the character drawn for medium lines at the upper right corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_UPPER_RIGHT** or **LDMUR**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_MEDIUM_VERTICAL

The LD_MEDIUM_VERTICAL statement defines the character drawn for medium vertical lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_MEDIUM_VERTICAL** or **LDMV**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_BEGIN

The LD_BOLD_BEGIN statement specifies the sequence sent to start a bold line. If omitted, no sequence needs to be sent to start a bold line.

The format is:

> **LD_BOLD_BEGIN** or **LDBB**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_CROSS

The LD_BOLD_CROSS statement defines the character drawn for bold lines at the point where the lines cross. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_CROSS** or **LDBC**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_DOWN_T

The LD_BOLD_DOWN_T statement specifies the sequence that defines the character drawn for bold lines at the meeting point of the horizontal line and a line that originates at the horizontal line and goes upward. The following character appears on the screen:

⊥

If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_DOWN_T** or **LDBDT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_END

The LD_BOLD_END statement specifies the sequence sent to end a bold line. If omitted, no sequence needs to be sent to end a bold line.

The format is:

> **LD_BOLD_END** or **LDBE**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_HORIZONTAL

The LD_BOLD_HORIZONTAL statement defines the character drawn for horizontal bold lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_HORIZONTAL** or **LDBH**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_LEFT_T

The LD_BOLD_LEFT_T statement specifies the sequence that defines the character drawn for bold lines at the meeting point of the vertical line and a line that originates at the vertical line and goes to the right. The following character appears on the screen:

├

If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_LEFT_T** or **LDBLT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_LOWER_LEFT

The LD_BOLD_LOWER_LEFT statement defines the character drawn for bold lines at the lower left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_LOWER_LEFT** or **LDBLL**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_LOWER_RIGHT

The LD_BOLD_LOWER_RIGHT statement defines the character drawn for bold lines at the lower right corner of a rectangle. If omitted, the terminal does not have this capability.

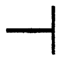
The format is:

> **LD_BOLD_LOWER_RIGHT** or **LDBLR**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_RIGHT_T

The LD_BOLD_RIGHT_T statement specifies the sequence that defines the character drawn for bold lines at the meeting point of the vertical line and a line that originates at the vertical line and goes to the left. The following character appears on the screen:



If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_RIGHT_T** or **LDBRT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_UP_T

The LD_BOLD_UP_T statement specifies the sequence that defines the character drawn for bold lines at the meeting point of the horizontal line and a line that originates at the horizontal line and goes downward. The following character appears on the screen:

┬

If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_UP_T** or **LDBUT**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_UPPER_LEFT

The LD_BOLD_UPPER_LEFT statement defines the character drawn for bold lines at the upper left corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_UPPER_LEFT** or **LDBUL**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_UPPER_RIGHT

The LD_BOLD_UPPER_RIGHT statement defines the character drawn for bold lines at the upper right corner of a rectangle. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_UPPER_RIGHT** or **LDBUR**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

LD_BOLD_VERTICAL

The LD_BOLD_VERTICAL statement defines the character drawn for vertical bold lines. If omitted, the terminal does not have this capability.

The format is:

> **LD_BOLD_VERTICAL** or **LDBV**
> **OUT=list of integer, keyword,** or **string**

The OUT (O) parameter specifies the character sequence sent to the terminal. This parameter is required.

Defining Functions and Key Labels for EDIT_FILE

You have the following choices for defining the programmable function keys for the EDIT_FILE utility:

1. Let EDIT_FILE default to assigning the subcommands and labels associated with the programmable function keys. The defaults used are listed in table 2-1.

2. Use a separate APPLICATION_STRING statement to define each programmable function key.

3. Use the SET_FUNCTION_KEY subcommand in the editor prolog file to define each programmable function key.

Using the APPLICATION_STRING statement is more efficient than using the SET_FUNCTION_KEY subcommand in an editor prolog. However, not all function keys used by EDIT_FILE can be assigned with the APPLICATION_STRING statement. In particular, the shifted and unshifted definitions for the DATA, DOWN, EDIT, FWD, BKW, BACK, HELP, STOP, UNDO, and UP keys must be defined through the SET_FUNCTION_KEY subcommand. If you have defined any of these keys for the terminal and you want to override the default definition assigned by EDIT_FILE for these keys, do the following:

1. Define the programmable function keys (function keys 1 through 16) through APPLICATION_STRING statements.

2. Create an editor prolog for the definition of these keys by the SET_FUNCTION_KEY subcommand.

Table 2-1. EDIT_FILE Defaults for Function Keys

Cap/Op	Value Used from Terminal Definition
InsCh	INSERT_CHAR with nonblank LABEL, or INSERT_MODE_BEGIN and INSERT_MODE_END with nonblank LABEL
DelCh	DELETE_CHAR with nonblank LABEL
Bkw	BKW with nonblank LABEL, or F1, or F-key with IN the same as BKW IN
First	BKW_S with nonblank LABEL, or F1_S, or F-key with IN the same as BKW_S IN
Fwd	FWD with nonblank LABEL, or F2, or F-key with IN the same as FWD IN
Last	FWD_S with nonblank LABEL, or F2_S, or F-key with IN the same as FWD_S IN
Back	BACK with nonblank LABEL, or F3, or F-key with IN the same as BACK IN
Help	HELP with nonblank LABEL, or F4, or F-key with IN the same as HELP IN
Undo	UNDO with nonblank LABEL, or F5, or F-key with IN the same as UNDO IN
Redo	UNDO_S with nonblank LABEL, or F5_S, or F-key with IN the same as UNDO_S IN. (Redo is not currently supported by EDIT_FILE.)
Quit	STOP with nonblank LABEL, or F6, or F-key with IN the same as STOP IN
Exit	STOP_S with nonblank LABEL, or F6_S, or F-key with IN the same as STOP_S IN
InsLn	INSERT_LINE_BOL or INSERT_LINE_STAY with nonblank LABEL

(Continued)

Table 2-1. EDIT_FILE Defaults for Function Keys *(Continued)*

Cap/Op	Value Used from Terminal Definition
DelLn	DELETE_LINE_BOL or DELETE_LINE_STAY with nonblank LABEL
Home	CURSOR_HOME with nonblank LABEL
OPS	The operations Copy, Move, Mark, Unmrk, MrkCh, MrkBx, Locate, LocNxt, LocAll, Width, Break, Join, and SkpEL cannot be defined through a TDU statement; they are always assigned programmable function keys.
ClrEL	ERASE_END_OF_LINE with nonblank LABEL
Middle	This operation cannot be defined through a TDU statement; it is always assigned a programmable function key.
Refrsh	ERASE_PAGE_HOME or ERASE_PAGE_STAY with nonblank LABEL
LinUp	UP with nonblank LABEL
LinDn	DOWN with nonblank LABEL
OPS	The operations Format, Center, InsWd, DelWd, InsBk, DelBk, Indent, and Dedent cannot be defined through a TDU statement; they are always assigned programmable function keys.

APPLICATION_STRING

The format of the APPLICATION_STRING statement is:

```
APPLICATION_STRING or APPS
   NAME=string
   OUT=string
```

The NAME (N) parameter specifies the character string that the application associates with the programmable function key. This parameter is required. NAME parameter values for user-defined applications must be in the document that describes the application. Values for the EDIT_FILE utility follow.

On a statement defining the function of a key, use the following rules to determine the value for the NAME parameter.

- For an unshifted key use:

  FSE_FUNCTION_

  followed by the number of the key. For example, the name of the function of unshifted programmable function key F8 is:

  FSE_FUNCTION_8

- For a shifted programmable function key use:

  FSE_FUNCTION_SHIFT_

  followed by the number of the key. For example, the name associated with shifted programmable function key F8 is:

  FSE_FUNCTION_SHIFT_8

  On a statement defining the label of a key, the entry is the name of the function of the key (as just described) followed by _LABEL.

- For the unshifted F8 key label use:

  FSE_FUNCTION_8_LABEL

- For the shifted F8 key label use:

  FSE_FUNCTION_SHIFT_8_LABEL

The OUT (O) parameter specifies the string associated with the value in the NAME parameter. It is sent to the application, which can use it any way it wants. This parameter is required. The OUT parameter can be continued on more than one line under the following conditions:

- Strings that would extend over more than one line must be broken into substrings that the system concatenates. Each substring must be complete on a single line.

- Variables must be complete on each line.

You can use variable names to define lengthy subcommands, as in the following example.

```
   f4a='write_file f=$local.t$.$boi,l=m'
   f4b='format_cybil_source i=$local.t$.$boi o=$local.t1$.$boi'
   f4c='delete_lines l=m'
   f4d='read_file f=$local.t1$ p=b'
   application_string name=('FSE_FUNCTION_4')..
    out=(f4a ';' f4b ';' f4c ';' f4d )
```

For user-defined applications, refer to the document that describes the application. Information for the EDIT_FILE utility follows.

When defining the function of a key, the string for the OUT parameter is the subcommand executed when the key is pressed. When you define the label of a key, the string is the label that appears on the screen. Refer to the NOS/VE File Editor manual listed in appendix B for both values.

Defining Functions and Key Labels for Applications Other Than EDIT_FILE

The full screen applications Debug, EDIT_CATALOG, EXPLAIN, IM/Quick, and Programming Environments define both the functions performed and labels assigned to programmable function keys through application menus. You can change the application menu if you wish to change either the function key or the label used by these applications. Application menus are described in the NOS/VE Object Code Management manual.

Appendixes

Glossary A

A

ANSI

American National Standards Institute.

ASCII

American Standard Code for Information Interchange.

Asynchronous

A type of terminal that has successive bits, characters, or events transmitted at variable intervals. In data transmission this is usually limited to a variable time interval between characters and is often known as start-stop transmission. Contrast with Synchronous.

B

Boolean

A kind of value that is evaluated as TRUE or FALSE.

Boolean Constant

A constant that represents a boolean (logical) value of TRUE or FALSE. One of the following names can be used to specify a boolean constant:

TRUE	FALSE
YES	NO
ON	OFF

C

Character

A letter, digit, space, or symbol that is represented by a code in one or more of the standard character sets.

It is also referred to as a byte when used as a unit of measure to specify block length, record length, and so forth.

A character can be a graphic character or a control character. A graphic character is printable; a control character is nonprintable and is used to control an input or output operation.

Command

A statement that initiates a specific operation on NOS/VE. A command name is recognized by the SCL interpreter if it appears as an entry in the command list.

Comment

A line or sequence of characters that is not interpreted or compiled and is for documentary purposes only.

Cycle Reference

The cycle of a permanent file to be accessed. A cycle reference can be either an unsigned integer or one of the following designators:

$HIGH

$LOW

$NEXT

D

Direct Cursor Addressing

The ability of the terminal to place the cursor immediately at any set of coordinates on the screen.

E

EDIT_FILE Utility

A NOS/VE command utility that provides an editor which enables you to edit files either by page in full screem mode or line by line.

F

File

An SCL element that specifies a temporary or permanent file, including its path and, optionally, a cycle reference (for permanent files). See also Path and Cycle Reference.

File Organization

Defines the way records are stored in a file. The available file organizations are sequential, byte-addressable, direct-access, and indexed-sequential.

File Position

The location in the file at which the next read or write operation will begin. A file that can be positioned is identified by specifying a path, an optional cycle reference (for permanent files), and an optional file position as follows:

path.cycle reference.file position

The file position designators are:

$ASIS	Leave the file in its current position.
$BOI	Position the file at the beginning-of-information.
$EOI	Position the file at the end-of-information.

See also Path and Cycle Reference.

FMU

File Management Utility

Function Key

A key on the terminal that, when pressed, performs a specified operation. The operation can be either defined by the software or built into the terminal.

I

Integer

A value representing one of the numbers 0, +1, -1, +2, -2, and so forth.

J

Job

A set of tasks executed for a user name. NOS/VE accepts interactive and batch jobs. In interactive mode, a job is usually the same as a terminal session.

Job Library List

Object libraries included in the program library list for each program executed in the job.

K

Keyword

A parameter value that has special meaning in the context of a particular parameter. For example, a parameter called COUNT might normally expect an integer but could be given the keyword ALL.

L

Load Module

A module reformatted for code sharing and efficient loading. When the user generates an object library, each object module in the module list is reformatted and written as a load module on the object library.

M

Module

A unit of text accepted as input by the loader, linker, or object library generator. See also Object Module and Load Module.

N

NOS/VE

Network Operating System, Virtual Environment.

O

Object Library

A file containing one or more load, SCL procedures, program description, message, and/or application modules and a dictionary to each module.

Object Module

A compiler-generated unit containing object code and instructions for loading the object code. It is accepted as input by the system loader and the CREATE_OBJECT_LIBRARY utility.

P

Parameter

A value list optionally preceded by and equated to a parameter name. For example:

```
parameter name = value list

    or

value list
```

Parameter Name

A name that uniquely identifies a parameter.

Path

In NOS/VE, a path specifies the location of a file in a catalog hierarchy. A general example of a path, from highest to lowest level in its hierarchy, is family name, user name (or master catalog name), subcatalog name(s), and file name. See also File, File Cycle, and Cycle Reference.

Permanent File

A mass storage file preserved by NOS/VE across job executions and system deadstarts. A permanent file has an entry in a permanent catalog. See also File.

Prolog

The SCL statement list that is executed at the beginning of each job.

R

Range

Value represented as two values separated by an ellipsis. The element is associated with the values from the first value through the second value. The first value must be less than or equal to the second value. For example:

value..value

S

SCL

See System Command Language.

SCU

See Source Code Utility.

Source Code Utility (SCU)

A NOS/VE command utility that stores, organizes, manipulates, and extracts units of text. It is a development tool for large systems or application development groups.

Source Library

A collection of decks on a file, with a header describing the collection, generated and manipulated by the Source Code Utility (SCU).

Statement

A combination of words and symbols.

String

A value that represents a sequence of characters.

Synchronous

A type of terminal that has successive bits, characters, or events transmitted at constant intervals. In data transmission this is usually limited to a constant time interval between characters. Contrast with Asynchronous.

System Command Language (SCL)

The block-structured interpretive language that provides the interface to the features and capabilities of NOS/VE. All commands and statements are interpreted by SCL before being processed by the system.

T

TDU

Terminal Definition Utility.

Terminal Definition File

The source file used in defining a terminal for use with a full-screen application.

V

Value

An expression or application value specified in a parameter list. Each value must match the defined kind of value for the parameter. Keywords, constants, and variable references are all values.

Value Element

A single value or a range of values represented by two values separated by an ellipsis. For example:

```
value

    or

value..value
```

See also Value, Value List, and Value Set.

Value List

A series of value sets separated by spaces or commas and enclosed in parentheses. If only one value set is given in the list, the parentheses can be omitted. For example:

```
(value set,value set,value set)

    or

value set
```

See also Value, Value Element, and Value Set.

Value Set

A series of value elements separated by spaces or commas and enclosed in parentheses. If only one value element is given in the set, the parentheses can be omitted. For example:

```
    (value element,value element,value element)

        or

    value element
```

See also Value, Value Element, and Value List.

W

Working Catalog

The catalog used if no other catalog is specified on a file reference. The initial working catalog within a job is the $LOCAL catalog.

Related Manuals B

All NOS/VE manuals and related hardware manuals are listed in table B-1. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
/explain
```

Ordering Printed Manuals

To order a printed Control Data manual, send an order form to:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

To obtain an order form or to get more information about ordering Control Data manuals, write to the above address or call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

Accessing Online Manuals

To access the online version of a printed manual, log in to NOS/VE and enter the online title on the EXPLAIN command (table B-1 supplies the online titles). For example, to see the NOS/VE Commands and Functions manual, enter:

```
/help manual=scl
```

The examples in some printed manuals exist also in the online Examples manual. To access this manual, enter:

```
/help manual=examples
```

When EXAMPLES is listed in the Online Manuals column in table B-1, that manual is represented in the online Examples manual.

Table B-1. Related Manuals

Manual Title	Publication Number	Online Manuals[1]
NOS/VE Site Manuals:		
CYBER 930 Computer System Guide to Operations Usage	60469560	
CYBER Initialization Package (CIP) Reference Manual	60457180	
Desktop/VE Host Utilities Usage	60463918	
MAINTAIN_MAIL[2] Usage		MAIM
NOS/VE Accounting Analysis System Usage	60463923	
NOS/VE Accounting and Validation Utilities for Dual State Usage	60458910	
NOS/VE LCN Configuration and Network Management Usage	60463917	
NOS/VE Network Management Usage	60463916	
NOS/VE Operations Usage	60463914	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

2. To access this manual, you must be the administrator for MAIL/VE.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Site Manuals (Continued):		
NOS/VE System Performance and Maintenance Volume 1: Performance Usage	60463915	
NOS/VE System Performance and Maintenance Volume 2: Maintenance Usage	60463925	
NOS/VE User Validation Usage	60464513	
NOS/VE User Manuals:		
EDIT_CATALOG Usage		EDIT_ CATALOG
EDIT_CATALOG for NOS/VE Summary	60487719	
Introduction to NOS/VE Tutorial	60464012	
NOS/VE Advanced File Management Tutorial	60486412	AFM_T

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
NOS/VE User Manuals (Continued):		
NOS/VE Advanced File Management Usage	60486413	AFM
NOS/VE Advanced File Management Summary	60486419	
NOS/VE Commands and Functions Quick Reference	60464018	SCL
NOS/VE File Editor Tutorial/Usage	60464015	EXAMPLES
NOS/VE Object Code Management Usage	60464413	OCM
NOS/VE Screen Formatting Usage	60488813	EXAMPLES
NOS/VE Source Code Management Usage	60464313	SCM and EXAMPLES
NOS/VE System Usage	60464014	EXAMPLES
NOS/VE Terminal Definition Usage	60464016	
Screen Design Facility for NOS/VE Usage	60488613	SDF

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
CYBIL Manuals:		
CYBIL for NOS/VE File Management Usage	60464114	EXAMPLES
CYBIL for NOS/VE Keyed-File and Sort/Merge Interfaces Usage	60464117	EXAMPLES
CYBIL for NOS/VE Language Definition Usage	60464113	CYBIL and EXAMPLES
CYBIL for NOS/VE Sequential and Byte-Addressable Files Usage	60464116	EXAMPLES
CYBIL for NOS/VE System Interface Usage	60464115	EXAMPLES

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
FORTRAN Manuals:		
FORTRAN Version 1 for NOS/VE Language Definition Usage	60485913	EXAMPLES
FORTRAN Version 1 for NOS/VE Quick Reference		FORTRAN
FORTRAN Version 2 for NOS/VE Language Definition Usage	60487113	EXAMPLES
FORTRAN Version 2 for NOS/VE Quick Reference		VFORTRAN
FORTRAN for NOS/VE Tutorial	60485912	FORTRAN_T
FORTRAN for NOS/VE Topics for FORTRAN Programmers Usage	60485916	
FORTRAN for NOS/VE Summary	60485919	
COBOL Manuals:		
COBOL for NOS/VE Summary	60486019	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
COBOL Manuals (Continued):		
COBOL for NOS/VE Tutorial	60486012	COBOL_T
COBOL for NOS/VE Usage	60486013	COBOL and EXAMPLES
Other Compiler Manuals:		
ADA for NOS/VE Usage	60498113	ADA
ADA for NOS/VE Reference Manual	60498118	EXAMPLES
APL for NOS/VE File Utilities Usage	60485814	
APL for NOS/VE Language Definition Usage	60485813	
BASIC for NOS/VE Summary Card	60486319	
BASIC for NOS/VE Usage	60486313	BASIC
LISP for NOS/VE Usage Supplement	60486213	
Pascal for NOS/VE Summary Card	60485619	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Other Compiler Manuals (Continued):		
Pascal for NOS/VE Usage	60485613	PASCAL and EXAMPLES
Prolog for NOS/VE Quick Reference	60486718	PROLOG
Prolog for NOS/VE Usage	60486713	
VX/VE Manuals:		
C/VE for NOS/VE Quick Reference		C
C/VE for NOS/VE Usage	60469830	
DWB/VX Introduction and User Reference Tutorial/Usage	60469890	
DWB/VX Macro Packages Guide Usage	60469910	
DWB/VX Preprocessors Guide Usage	60469920	
DWB/VX Text Formatters Guide Usage	60469900	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
VX/VE Manuals (Continued):		
VX/VE Administrator Guide and Reference Tutorial/Usage	60469770	
VX/VE An Introduction for UNIX Users Tutorial/Usage	60469980	
VX/VE Programmer Guide Tutorial	60469790	
VX/VE Programmer Reference Usage	60469820	
VX/VE Support Tools Guide Tutorial	60469800	
VX/VE User Guide Tutorial	60469780	
VX/VE User Reference Usage	60469810	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Data Management Manuals:		
DM Command Procedures Reference Manual	60487905	
DM Concepts and Facilities Manual	60487900	
DM Error Message Summary for DM on CDC NOS/VE	60487906	
DM Fundamental Query and Manipulation Manual	60487903	
DM Report Writer Reference Manual	60487904	
DM System Administrator's Reference Manual for DM on CDC NOS/VE	60487902	
DM Utilities Reference Manual for DM on CDC NOS/VE	60487901	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Information Management Manuals:		
IM/Control for NOS/VE Quick Reference	L60488918	CONTROL
IM/Control for NOS/VE Usage	60488913	
IM/Quick for NOS/VE Tutorial	60485712	
IM/Quick for NOS/VE Summary	60485714	
IM/Quick for NOS/VE Usage		QUICK
CDCNET Manuals:		
CDCNET Access Guide	60463830	CDCNET_ ACCESS
CDCNET Batch Device User Guide	60463863	CDCNET_ BATCH
CDCNET Commands Quick Reference	60000020	
CDCNET Configuration and Site Administration Guide	60461550	
CDCNET Diagnostic Messages	60461600	
CDCNET Conceptual Overview	60461540	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
CDCNET Manuals (Continued):		
CDCNET Network Analysis	60461590	
CDCNET Network Configuration Utility		NETCU
CDCNET Network Configuration Utility Summary Card	60000269	
CDCNET Network Operations	60461520	
CDCNET Network Performance Analyzer	60461510	
CDCNET Product Descriptions	60460590	
CDCNET Systems Programmer's Reference Manual Volume 1 Base System Software	60462410	
CDCNET Systems Programmer's Reference Manual Volume 2 Network Management Entities and Layer Interfaces	60462420	
CDCNET Systems Programmer's Reference Manual Volume 3 Network Protocols	60462430	
CDCNET Terminal Interface Usage	60463850	
CDCNET TCP/IP Usage	60000214	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Migration Manuals:		
Migration from IBM to NOS/VE Tutorial/Usage	60489507	
Migration from NOS to NOS/VE Tutorial/Usage	60489503	
Migration from NOS to NOS/VE Standalone Tutorial/Usage	60489504	
Migration from NOS/BE to NOS/VE Tutorial/Usage	60489505	
Migration from NOS/BE to NOS/VE Standalone Tutorial/Usage	60489506	
Migration from VAX/VMS to NOS/VE Tutorial/Usage	60489508	
Miscellaneous Manuals:		
Applications Directory	60455370	
CONTEXT Summary Card	60488419	
CYBER Online Text for NOS/VE Usage	60488403	CONTEXT
Control Data CONNECT User's Guide	60462560	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Miscellaneous Manuals (Continued):		
Debug for NOS/VE Quick Reference		DEBUG
Debug for NOS/VE Usage	60488213	
Desktop/VE for Macintosh Tutorial	60464502	
Desktop/VE for Macintosh Usage	60464503	
NOS/VE Diagnostic Messages Usage	60464613	MESSAGES
MAIL/VE Summary Card	60464519	
MAIL/VE Usage		MAIL_VE
Math Library for NOS/VE Usage	60486513	
NOS/VE Examples Usage		EXAMPLES
NOS/VE System Information		NOS_VE

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Miscellaneous Manuals (Continued):		
Programming Environment for NOS/VE Usage		ENVIRON-MENT
Programming Environment for NOS/VE Summary	60486819	
Professional Programming Environment for NOS/VE Quick Reference		PPE
Professional Programming Environment for NOS/VE Usage	60486613	
Remote Host Facility Usage	60460620	
Hardware Manuals:		
CYBER 170 Computer Systems Models 825, 835, and 855 General Description Hardware Reference	60459960	
CYBER 170 Computer Systems, Models 815, 825, 835, 845, and 855 CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, and 860 Codes Booklet	60458100	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Hardware Manuals (Continued):		
CYBER 170 Computer Systems, Models 815, 825, 835, 845, and 855 CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, and 860 Maintenance Register Codes Booklet	60458110	
HPA/VE Reference	60461930	
Virtual State Volume II Hardware Reference	60458890	
7021-31/32 Advanced Tape Subsystem Reference	60449600	
7221-1 Intelligent Small Magnetic Tape Subsystem Reference	60461090	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

Character Set C

ASCII Character Set

This appendix lists the ASCII character set (refer to table C-1).

NOS/VE supports the American National Standards Institute (ANSI) standard ASCII character set (ANSI X3.4-1977). NOS/VE represents each 7-bit ASCII code in an 8-bit byte. These 7 bits are right justified in each byte. For ASCII characters, the eighth or leftmost bit is always zero. However, in NOS/VE the leftmost bit can also be used to define an additional 128 characters.

If you want to define additional non-ASCII characters, be certain that the leftmost bit is available in your current working environment. The full screen applications (such as the EDIT_FILE utility, the EDIT_CATALOG utility, and the programming language environments) already use this bit for special purposes. Therefore, these applications accept only the standard ASCII characters. In applications in which the leftmost bit is not used, however, you are free to use it to define the interpretation of each character as you wish.

Table C-1. ASCII Character Set

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
000	00	000	NUL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign
036	24	044	$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
040	28	050	(	Opening parenthesis
041	29	051	)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus
044	2C	054	,	Comma
045	2D	055	−	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equals
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	[	Opening bracket
092	5C	134	\	Reverse slant
093	5D	135	]	Closing bracket
094	5E	136	^	Circumflex
095	5F	137	_	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w

(Continued)

Table C-1. **ASCII Character Set** (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174	|	Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

VT220 Terminal Definition File D

This appendix gives an example of a terminal definition for the VT220 terminal. The example shows how terminal definitions are set up. Comments within the file are in quotation marks. This example is for reference only. To obtain released versions of actual terminal definition files, refer to Creating a Terminal Definition in chapter 1 of this manual.

The VT220 definition uses the following characteristics:

- The VT220 does not provide all three CDC line drawing densities. The VT220 line drawing set is used for fine density, and with alternate intensity for medium density. Blanks with inverse video are used for bold density.
- The VT220 has physical function keys named F6 through F20, which are translated into terminal definition keys F1 through F15. The LABEL parameters use the F6 through F20 notation to tell the keyboard user which keys to press.
- Protected fields are not used.
- The keypad keys can be converted into a type of function key.
- The shifted function keys F6 through F20 can be dynamically loaded. The CDC-supplied terminal definition uses this capability to provide keys for local editing functions of insertion and deletion, plus a few high-frequency standard functions, as shown in the following table. The keyboard user does not need to press the Return key to use these shifted keys.

VT220 Key	CDC Terminal Definition Statement
Shifted F6	INSERT_LINE_BOL
Shifted F7	DELETE_LINE_BOL
Shifted F8	INSERT_CHAR
Shifted F9	DELETE_CHAR
Shifted F10	INSERT_MODE_BEGIN
Shifted F11	INSERT_MODE_END
Shifted F12	ERASE_END_OF_LINE
Shifted F13	ERASE_PAGE_HOME
Shifted F14	BACK
Shifted Help (F15)	HELP

VT220 Key	CDC Terminal Definition Statement
Shifted Do (F16)	CURSOR_HOME
Shifted F17	BKW_S
Shifted F18	BKW
Shifted F19	FWD
Shifted F20	FWD_S

- The CDC programmable function keys all require the user to press the physical Return key to get a response, and are mapped as shown in the following table.

VT220 Key	CDC Terminal Definition Statement
F6 through F20	F1 through F15
Keypad Enter	F16
Keypad 1 through 9	F1_S through F9_S
Keypad 0	F10_S
Keypad PF1 through PF4	F11_S through F14_S
Keypad Dash	F15_S
Keypad Comma	F16_S

```
"   TERMINAL DEFINITION FILE FOR DIGITAL VT220 TERMINAL                        "

"   VARIABLES                                                                  "
prefix              = (1B(16) 5B(16))
fkey                = (1B(16) 4F(16))
escape              = (1B(16))
clear_home          = (prefix 32(16) 4A(16))
clear_all_tabs      = (prefix '3g')
ansi_mode           = (escape '<')
vt100_mode          = (prefix '61"p')
vt220_mode          = (prefix '62;1"p')
designate_ascii_g0  = (escape '(B')
designate_graph_g1  = (escape ')0')
select_g0           = (0F(16))
application_keypad  = (escape '=')
numeric_keypad      = (escape '>')
autowrap_off        = (prefix '?7l')
autowrap_on         = (prefix '?7h')
set_to_24x80        = (prefix 3F(16) 33(16) 6C(16))
set_to_24x132       = (prefix 3F(16) 33(16) 68(16))
start_alternate     = (prefix 31(16) 6D(16))
start_inverse       = (prefix '7' 6D(16))
start_underline     = (prefix 34(16) 6D(16))
normal_attributes   = (prefix 'm')
stop_alternate      = (prefix '22m')
stop_inverse        = (prefix '27m')
stop_underline      = (prefix '24m')

start_keyload       = (esc 'P0;1|')
stop_keyload        = (esc '\')
load_f6             = ('17/1b5b4c')          " insert line "
load_f7             = ('18/1b5b4d')          " delete line "
load_f8             = ('19/1b5b40')          " insert blank character "
load_f9             = ('20/1b5b50')          " delete character "
load_f10            = ('21/1b5b3468')        " start insert mode "
load_f11            = ('23/1b5b346c')        " stop insert mode "
load_f12            = ('24/1b5b4b')          " clear to end of line "
load_f13            = ('25/1b5b324a0d')      " clear screen "
load_f14            = ('26/1b5b3939397e0d')  " Back "
load_help           = ('28/1b5b32387e0d')    " Help "
load_do             = ('29/1b5b48')          " Do=home "
load_f17            = ('31/1b5b3939387e0d')  " First "
load_f18            = ('32/1b5b357e0d')      " Bkw duplicates PrevScreen "
load_f19            = ('33/1b5b367e0d')      " Fwd duplicates NextScreen "
load_f20            = ('34/1b5b3939377e0d')  " Last "
load_all_keys       = (start_keyload load_f6 ';' load_f7 ';' load_f8 ';' ..
  load_f9 ';' load_f10 ';' load_f11 ';' load_f12 ';' load_f13 ';' load_f14 ..
  ';' load_help ';' load_do ';' load_f17 ';' load_f18 ';' load_f19 ';' ..
  load_f20 stop_keyload)

"   MODEL NAME AND COMMUNICATION TYPE                                          "
model_name          value = 'dec_vt220'
communications      type  = asynch
application_string  name  = 'vt100_scrolling' out = 'true'

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

"   CURSOR POSITIONING INFORMATION                                             "
cursor_pos_encoding       bias  = (1)   type = ansi_cursor
cursor_pos_column_first   value = FALSE
cursor_pos_column_length  value = (0)
cursor_pos_row_length     value = (0)
cursor_pos_begin          out   = (prefix)
cursor_pos_second         out   = (3B(16))
cursor_pos_third          out   = (48(16))
```

```
"   CURSOR MOVEMENT INFORMATION                                               "
cursor_home              inout = (prefix 48(16))      label='shift-do'
cursor_up                inout = (prefix 41(16))
cursor_down              inout = (prefix 42(16))
cursor_left              inout = (prefix 44(16))
cursor_right             inout = (prefix 43(16))

"   CURSOR BEHAVIOR (for cursor movement keys)                                "
move_past_right          type  = stop_next
move_past_left           type  = stop_next
move_past_top            type  = stop_next
move_past_bottom         type  = stop_next

"   CURSOR BEHAVIOR (for character keys)                                      "
char_past_right          type  = stop_next
char_past_left           type  = stop_next
char_past_last_position  type  = stop_next

"   TERMINAL ATTRIBUTES                                                       "
clears_when_change_size  value = TRUE
function_key_leaves_mark value = 1
has_hidden               value = FALSE
has_protect              value = FALSE
home_at_top              value = TRUE
multiple_sizes           value = TRUE
tabs_to_home             value = FALSE
tabs_to_tab_stops        value = TRUE
tabs_to_unprotected      value = FALSE

"   SCREEN SIZES                                                              "
set_size        rows = 24 columns = 80   out = (set_to_24x80)
set_size        rows = 24 columns = 132  out = (set_to_24x132)

"   SCREEN AND LINE MODE TRANSITION                                           "
set_screen_mode     out = (ansi_mode vt220_mode clear_all_tabs  ..
     designate_ascii_g0 designate_graph_g1 select_g0 autowrap_off  ..
     load_all_keys application_keypad)

set_line_mode       out = (vt100_mode ansi_mode clear_all_tabs  ..
     designate_ascii_g0 designate_graph_g1 select_g0 numeric_keypad ..
     autowrap_on)

"et_screen_mode     out = (1B(16) 3C(16) clear_all_tabs ..
"     1B(16) 28(16) 42(16) 1B(16) 29(16) 30(16) 0F(16) 1B(16) ..
"     3D(16) prefix '?71' vt220_mode)

"et_line_mode     out = (1B(16) 3C(16) clear_all_tabs ..
"     1B(16) 28(16) 42(16) 1B(16) 29(16) 30(16) 0F(16) 1B(16) ..
"     3E(16) prefix '?7h')

"   TERMINAL CAPABILITIES                                                     "
delete_char         inout = (prefix 50(16))     label='shift-f9'
delete_line_bol     inout = (prefix 4D(16))     label='shift-f7'
erase_end_of_line   inout = (prefix 4B(16))     label='shift-f12'
erase_line_stay     inout = (prefix 32(16) 4B(16))
erase_page_home     inout = (clear_home)        label='shift-f13'
insert_char         inout = (prefix 40(16))     label='shift-f8'
insert_line_bol     inout = (prefix 4C(16))          label='shift-f6'
insert_mode_begin   inout = (prefix 34(16) 68(16))  label='shift-f10'
insert_mode_end     inout = (prefix 34(16) 6C(16))  label='shift-f11'
tab_forward         inout = (09(16))
tab_clear_all       inout = (clear_all_tabs)
tab_set             inout = (1B(16) 48(16))

"   MISCELLANEOUS TERMINAL SEQUENCES                                          "
bell_nak            out = (bel)
backspace           in = bs
```

```
"   PROGRAMMABLE FUNCTION KEY INPUT INFORMATION                          "
f1        in = (prefix '17~')         label='f6'
f2        in = (prefix '18~')         label='f7'
f3        in = (prefix '19~')         label='f8'
f4        in = (prefix '20~')         label='f9'
f5        in = (prefix '21~')         label='10'
f6        in = (prefix '23~')         label='11'
f7        in = (prefix '24~')         label='12'
f8        in = (prefix '25~')         label='13'
f9        in = (prefix '26~')         label='14'
f10       in = (prefix '28~')         label='He'
f11       in = (prefix '29~')         label='Do'
f12       in = (prefix '31~')         label='17'
f13       in = (prefix '32~')         label='18'
f14       in = (prefix '33~')         label='19'
f15       in = (prefix '34~')         label='20'

f16       in = (fkey 'M')             label='ke'

f1_s      in = (fkey 'q')             label='k1'
f2_s      in = (fkey 'r')             label='k2'
f3_s      in = (fkey 's')             label='k3'
f4_s      in = (fkey 't')             label='k4'
f5_s      in = (fkey 'u')             label='k5'
f6_s      in = (fkey 'v')             label='k6'
f7_s      in = (fkey 'w')             label='k7'
f8_s      in = (fkey 'x')             label='k8'
f9_s      in = (fkey 'y')             label='k9'
f10_s     in = (fkey 'p')             label='k0'

f11_s     in = (fkey 'P')             label='p1'
f12_s     in = (fkey 'Q')             label='p2'
f13_s     in = (fkey 'R')             label='p3'
f14_s     in = (fkey 'S')             label='p4'

f15_s     in = (fkey 'm')             label='k-'
f16_s     in = (fkey 'l')             label='k,'

"         in = (prefix '2~')          label='IH'   "
"         in = (prefix '1~')          label='Fi'   "
"         in = (prefix '3~')          label='Re'   "
"         in = (prefix '4~')          label='Se'   "

"   CDC STANDARD FUNCTION KEY INPUT INFORMATION                          "
next      in = 13             label = 'RETURN'
next_s    in = ()
bkw       in = (prefix '5~')          label='PS'
fwd       in = (prefix '6~')          label='NS'
back      in = (prefix '999~') label='shift-f14'
undo      in = (prefix '21~') label='10'
help      in = (prefix '28~') label='shift-help'
stop      in = (prefix '23~') label='F6'
bkw_s     in = (prefix '998~') label='  Shift-F17'
fwd_s     in = (prefix '997~') label='  Shift-F20'
undo_s    in = (fkey 'u') label='  Shift-F5'
stop_s    in = (fkey 'v') label='  Shift-F6'
down      in = ()
down_s    in = ()
up        in = ()
up_s      in = ()
edit      in = ()
edit_s    in = ()
data      in = ()
data_s    in = ()
```

```
"   TERMINAL VIDEO ATTRIBUTES                                                  "
alt_begin           out = (start_alternate)
alt_end             out = (stop_alternate)
blink_begin         out = (prefix 35(16) 6D(16))
blink_end           out = (prefix '25m')
inverse_begin       out = (start_inverse)
inverse_end         out = (stop_inverse)
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 = (start_inverse)
italic_end          out = (stop_inverse)

"   LINE DRAWING CHARACTER SPECIFICATION                                       "
ld_fine_begin            out = 0E(16)
ld_fine_end              out = 0F(16)
ld_fine_horizontal       out = 71(16)
ld_fine_vertical         out = 78(16)
ld_fine_upper_left       out = 6C(16)
ld_fine_upper_right      out = 6B(16)
ld_fine_lower_left       out = 6D(16)
ld_fine_lower_right      out = 6A(16)
ld_fine_up_t             out = 77(16)
ld_fine_down_t           out = 76(16)
ld_fine_left_t           out = 74(16)
ld_fine_right_t          out = 75(16)
ld_fine_cross            out = 6E(16)
ld_medium_begin          out = (0E(16) start_alternate)
ld_medium_end            out = (0F(16) stop_alternate)
ld_medium_horizontal     out = 71(16)
ld_medium_vertical       out = 78(16)
ld_medium_upper_left     out = 6C(16)
ld_medium_upper_right    out = 6B(16)
ld_medium_lower_left     out = 6D(16)
ld_medium_lower_right    out = 6A(16)
ld_medium_up_t           out = 77(16)
ld_medium_down_t         out = 76(16)
ld_medium_left_t         out = 74(16)
ld_medium_right_t        out = 75(16)
ld_medium_cross          out = 6E(16)
ld_bold_begin            out = (0E(16) start_inverse)
ld_bold_end              out = (0F(16) stop_inverse)
ld_bold_horizontal       out = 71(16)
ld_bold_vertical         out = 78(16)
ld_bold_upper_left       out = 6C(16)
ld_bold_upper_right      out = 6B(16)
ld_bold_lower_left       out = 6D(16)
ld_bold_lower_right      out = 6A(16)
ld_bold_up_t             out = 77(16)
ld_bold_down_t           out = 76(16)
ld_bold_left_t           out = 74(16)
ld_bold_right_t          out = 75(16)
ld_bold_cross            out = 6E(16)

"   DEFAULT KEY DEFINITIONS FOR THE FULL SCREEN EDITOR                         "


"   END OF TERMINAL DEFINITION FILE FOR DIGITAL VT220 TERMINAL                 "
```

Index

[illegible]

Index

Comments (continued from other side)

'lease fold on dotted line;
e[illegible]dges with tape only.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First-Class Mail Permit No. 8241 Minneapolis, MN

POSTAGE WILL BE PAID BY ADDRESSEE

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

Terminal Definition **60464016 D**

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

Who are you?	**How do you use this manual?**
☐ Manager	☐ As an overview
☐ Systems analyst or programmer	☐ To learn the product or system
☐ Applications programmer	☐ For comprehensive reference
☐ Operator	☐ For quick look-up
☐ Other ____________	

What programming languages do you use? ____________

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

Yes	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?

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

Would you like a reply? ☐ Yes ☐ No

From:

Name	Company
Address	Date
	Phone

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

Terminal Definition Statement Index

This index lists alphabetically the terminal definition statements described in this manual and the page on which each is described.