MS320 **PROGRAMMING REFERENCE MANUAL**

Version 4.01
December 2000

Trademarks are property of their respective holders.

Minisoft
1024 First Street, Snohomish WA
USA

Phone: 800-682-0200
FAX 360-563-2923
E-Mail Sales: sales@minisoft.com
E-Mail Technical Support: support@minisoft.com
http:\\www.minisoft.com

TABLE OF CONTENTS

COMMAND LANGUAGE

OVERVIEW

The Emulation Command Language (ECL) is a powerful command/script language that is similar to DCL, Digital's Command Language for VAX/VMS.

The ability to execute emulator commands from command files allows both simple and complex tasks to be automated. Some of the tasks that can be easily automated with command files are:

- Dialing and login
- File transfer
- Management of host programs
- Data logging and analysis
- Designing a menu driven user interface for host applications

1.1 COMMAND SYNTAX

Emulator commands appear in uppercase letters (e.g., WRITE HOST). The standard syntax is:

COMMAND /OPTION(S) argument(s)

Note: Arguments shown in brackets, [ ], are optional.

A command may be abbreviated to the minimum number of characters required to make it non-ambiguous. Multiple command arguments are separated by spaces.

All options begin with a slash (/). Options may be used anywhere in the command.

Examples: **SEND /FILTER TEXT**

SEND TEXT /FILTER

Both forms of the send command are valid.

If the argument is a string of characters, the options must immediately follow the command. Character string arguments (referred to as strings) must be enclosed in quotation marks.

Example: **DISPLAY/NOCR "Hello there"**

This example shows the use of an option with a string argument. The option directly follows the command, and the string (Hello there) is enclosed in quotes.

1.2 COMMAND EXECUTION

Emulator commands can be executed from:

- The command line prompt
- A keyboard or mouse definition
- The host computer
- A command file (see the Executing Command Files section)

1.2.1 Command Line Execution

To execute a command from the command line:

1) Click on ***Execute - Command Line***, the C> button on the CMD Toolbar, or press **CMD** (default is Alt C). The `CMD>` prompt displays.
2) Enter the command or command file specification at the command prompt.

Example 1: `CMD>`**SET HOST /DISCONNECT**

Disconnects the currently connected port.

1.2.1.1 Entering Multiple Commands

A series of commands can be given by entering interactive command mode. In interactive mode, the command prompt reappears after each command is executed. The INTERACTIVE command enters interactive mode. To terminate interactive mode, use the ENDINTERACTIVE command.

1.2.2 Executing from the Host

Emulator commands may be executed by the host using a DCS private control sequence.

$^{C}S_{I}$5|Command String$^{S}_{T}$

Note: $^{C}S_{I}$ and ^{S}T are 8-bit characters that can only be used on systems that support full 8-bit characters. $^{E}S_{C}$ [is the 7-bit equivalent of $^{C}S_{I}$. $^{E}S_{C}$ \ is the 7-bit equivalent of ^{S}T .

Example: **$^{C}S_{I}$5|SET HOST /DISCONNECT$^{S}_{T}$** or
$^{E}S_{C}$[5|SET HOST /DISCONNECT$^{E}S_{C}$

These commands are used to disconnect the currently connected port..

1.3 COMMAND FILES

Command files are text files that contain emulator commands. Command files are useful for automating tasks such as transferring files, logging on, and defining keyboard configurations. However, command files are not limited to these functions. Chapter 8 (Command File Programming) covers more advanced programming topics.

A command file executes each emulator command in sequence. Emulator command files execute from:

- The command line prompt
- A key definition
- The host computer
- The modem dialer
- A command file

1.3.1 Specifying a Command File

Prefixing a filename with an at symbol (@) tells the emulator to expect a command file. If the filename does not include an extension, the emulator automatically appends .ECF to the filename.

The default filename extension of .ECF may be overridden by specifying an extension with the command file name. A command file name can also include a path specification.

Command files can be executed using a search path. Click on ***Setup - General - Directories*** to set the command file default directory (search path).

1.3.2 Default Command File

A command file can be executed automatically when the emulator loads by entering the name in the Command File field in the Session Manager's Properties dialog box. Do not enter the @ symbol as part of the name, or an extension - the default .ECF, is assumed.

1.3.3 Command Line Execution

A command file can be executed at the CMD> prompt any time you are in the emulator.

1) Click on ***Execute - Command Line***, the C> button on the CMD Toolbar, or press **CMD** (default is Alt C). The CMD> prompt displays.
2) Type the @ followed by the name of the command file.
3) Press Return or click the checkmark icon. The command file executes.

Example: CMD>**@LOGIN**

Executes a command file named LOGIN.ECF.

1.3.4 Executing from the Host

An emulator command file can execute from the host computer system through a DCS Private control sequence.

$^{C}S_{I}5|@command\ file\ specification^{S}_{T}$

Note: $^{C}S_{I}$ and ^{S}T are 8-bit characters. They can only be used on systems that support full 8-bit characters. $^{E}S_{C}$á[is the 7-bit equivalent of $^{C}S_{I}$. $^{E}S_{C}$ \ is the 7-bit equivalent of ^{S}T.

Example: **$^{C}S_{I}5|@MENU^{S}T$** or
$^{E}S_{C}[5|@MENU^{E}S_{C}\backslash$

The host uses these commands in programs, script or command files to run MENU.ECF.

1.3.5 Nested Command Files

To specify a command file from within a command file, precede the command filename with the @ symbol. After a nested command file is completed, control returns to the next line of the calling command file.

1.3.6 Comments

Comments are used in command files to document the purpose of the file and each emulator command. Comments are prefixed with the exclamation point (!). Any data to the right of the exclamation point is ignored.

Example:
```
! This command file logs onto a VAX/VMS system and
! changes to the TEST directory.
WAIT “Username:”                    ! wait for host prompt
WRITE HOST “USER”                   ! send username to host
WAIT “Password:”                    ! wait for host prompt
WRITE HOST “USER_TEST”              ! send password to host
                                    ! change to test directory
WRITE HOST “SET DEF [.TEST]”
EXIT                                ! exit command file
```

Comments are used to clearly state the purpose of the file and describe each line of the command file.

1.4 ABORTING COMMANDS

To abort emulator commands and/or command file execution, click ***Execute - Abort***, or click on the Abort button.

1.5 EMULATOR COMMAND LIST

Table 1-1 Emulator Command List

Command	Function
BREAK	Send a communications break
CAPTURE	Captures text to a file
CLOSE	Close a file
CLS	Clear screen (short form)
CONTINUE	Resume execution of next command
DDE ADVISE	Create Advise Data Link
DDE CONNECT	Connect a client and server application
DDE DISCONNECT	Disconnect the specified conversation
DDE DISCONNECTALL	Disconnect all conversations
DDE EXECUTE	Send commands to the server to be executed
DDE POKE	Send a data item value to the server
DDE REQUEST	Request the value of a data item from the server
DDE TOPICS	Compile a list of active server applications and topics
DDE UNADVISE	Delete an Advise Data Link
DELAY	Delay specified time
DELETE SYMBOL	Delete symbol(s)
DISPLAY	Output data (emulator to screen)
DOS	Execute DOS command
DROPDTR	Drop Data Terminal Ready (DTR)
EMULATE	Enter Emulation mode
ENDINTERACTIVE	End interactive command mode
ERASE SCREEN	Erase the screen
EXIT	Exit to DOS
FILE	Perform a file transfer
FLUSH	Flush receive buffer
GOSUB	Execute a subroutine within a command file
GOTO	Go to a command file label
HELP	Display emulator Help
IF	Test condition
INQUIRE	Prompt for input
INTERACTIVE	Enter interactive command mode
KERMIT	Enter Kermit mode
KERMIT BYE	Logout from the host and exit emulator mode
KERMIT CONNECT	Return to emulation mode
KERMIT DOS	Execute DOS command
KERMIT END	End Kermit Server session
KERMIT EXIT	Exit to Windows
KERMIT FINISH	Tell server to exit
KERMIT GET	Receive files from server
KERMIT HELP	Display Kermit help

Table 1-1 Emulator Command List (cont'd)

Command	Function
KERMIT LOGOUT	Tell server to logout
KERMIT RECEIVE	Non-server receive file
KERMIT SEND	Send file to server
LOG	Create a log file of session
ON ABORT	Set condition for ON ABORT
ON DEVICE_ERROR	Set condition for ON DEVICE_ERROR
ON DISCONNECT	Set condition for ON DISCONNECT
ON error_severity	Set condition for ON error levels
OPEN	Open a file
PRINT EJECT	Eject printer page
PRINT ON/OFF	Print on/off
PRINT SCREEN	Print the text screen
PRINT SCROLLBACK	Prints text in scrollback memory plus the screen contents
QUIT	Exit emulate mode
READ	Read a string from the host or file
READ HOST	Read an ASCII record from host into the specified symbol.
READ SCREEN	Read screen text into symbol
REPLAY	Replay an emulator Log file
RETURN	Return from a GOSUB command
SCAN	Display the key names
SEND	Send ASCII text file to host
SESSION	Start a session defined in the Session Manager
SET ABORT	Set Abort key checking
SET CDELAY	Set delay for sending characters
SET [NO]DDEAUTOINITIALIZE	Set DDE auto initialize
SET [NO]DDEAPPPENDINSTANCE	Set DDE append instance
SET DDECLIENTTIMEOUT	Set timeout value for DDE client commands
SET DDEERVERNAME	Set DDE server name
SET DEVICE_ERROR	Set device error checking
SET DISCONNECT	Set disconnect checking
SET EOF	Set the End of File character
SET HOST	Create a connection to a remote node
SET LDELAY	Set delay for sending lines
SET MESSAGE	Set message control
SET ON	Set error checking
SET TERMINAL	Set terminal characteristics
SET TURNAROUND	Set a turnaround character
SET VERIFY	Set verify mode
SHOW SYMBOL	Display local and global symbol values
STOP	Terminate execution of all command files
WAIT	Wait for a host string
WIN	Launch Windows application
WP5 ON/OFF	Enable/Disable WordPerfect 5.x mode
WRITE	Write a string to the host or file

1.5.1 Emulator Command Descriptions

BREAK

BREAK (no arguments)

Sends a 200 millisecond communications break to the communications port.

Valid options:

/LONG

Sends a long (3.5 second) break.

CAPTURE

CAPTURE filename

Captures the data received by the emulator to a file. The data is saved as it appears on the screen in pure text format.

Valid options:

/APPEND

Open a capture file and appends the text data to the end of file. If no file exists, one is created.

/CLOSE

Close the previously opened capture file. The filename is not required.

/OPEN

Create a capture file.

/OVERWRITE

Open a capture file and overwrite any old copies. If no file exists, one is created.

/SCREEN

Write the current screen contents to the previously opened capture file. This command formats the data with spaces exactly as it appears on the screen. None of the terminal escape sequences used to format the screen are written to the capture file.

Example 1**: CAPTURE TEST**

Creates a capture file TEST.TXT. If TEST.TXT already exists, an error occurs.

Example 2:
```
CAPTURE TEST/OPEN
CAPTURE/SCREEN
WAIT “END OF DATA”
CLOSE ERRORS
CAPTURE/CLOSE
```
Opens a capture file, saves the screen, and then captures data until the string “END OF DATA” appears.

Example 3:
```
CAPTURE/CLOSE
```
Closes the log file.

Example 4:
```
CAPTURE/OVER TEST
```
Opens TEST.TXT and overwrites any old copies.

Example 5:
```
WRITE HOST “MAIL”
WRITE HOST “READ”
CAPTURE MAIL
WRITE HOST “EXIT”
```
Captures a host mail message into a MAIL.TXT file.

CLOSE

CLOSE logical-name[:]

Where: **logical-name** is a DOS file logical assigned by the OPEN command.

Closes the logical name previously opened with the OPEN command. If the CLOSE command is not issued, the logical name is closed upon exiting the emulator.

Valid options:

/ERROR=label

Process continues at the label if an error occurs.

Example:
```
!Get user input into DATE
INQUIRE DATE “Enter current date and time: ”
OPEN/WRITE FILE DATA.LOG          ! Open PC file DATA.LOG
WRITE FILE DATE                   ! Write DATE into FILE
CLOSE FILE                        ! Close PC file
```
Places a date and time stamp on a log file by opening the PC file DATA.LOG, writing the date, and closing the file. DATA.LOG can be added later to the LOG/APPEND command.

Related topics: OPEN

CLS

CLS (no arguments)

Clears the screen. CLS is the short form of the ERASE SCREEN command.

Example: **WRITE HOST "ls"**
DELAY 3
INQUIRE FILENAME "Enter name of file to delete: "
WRITE HOST "rm''FILENAME'"
CLS

This Unix example lists the contents of a directory, removes the specified file from that directory, and clears the screen.

Related topics: ERASE SCREEN

CONTINUE

CONTINUE (no arguments)

Resumes execution on the next line of a command file. Used with the ON command to ignore error conditions.

Example: **ON ERROR THEN CONTINUE**

If an error occurs, the command continues at the next line.

DDE ADVISE

DDE ADVISE variable1 "item name" variable2

Where: **Variable1** is the conversation number returned by an earlier DDE CONNECT command.

"Item name" is a string expression that tells the server what data item to monitor.

Variable2 specifies the variable to receive the new data item value. Variable2 changes whenever the value of the data item in the server application changes.

Creates an Advise Data Link between the emulator (the client) and the server application. The value of the emulator variable is updated whenever the specified item's value in the server application changes. An Advise Data Link can be removed with the DDE UNADVISE command. All Advise Data Links associated with a conversation are removed when the conversation is disconnected.

Example: **DDE ADVISE 'CONV' "COUNT" RESULT**

Assumes that CONV refers to a conversation with another copy of the emulator as the DDE server using the ECL topic. An Advise Data Link is created so that when the DDE server's variable COUNT changes, the new value is assigned to the variable RESULT in the DDE client copy of the emulator.

Related topics: DDE UNADVISE

DDE CONNECT

DDE CONNECT “service name” “topic name” variable

Where: **“Service name”** is a string expression that corresponds to a DDE server application name. An empty string (“”) can be used as a wildcard to find all DDE server applications.

“Topic name” is a string expression that corresponds to the desired DDE conversation topic. An empty string (“”) can be used as a wildcard to find the DDE conversation topics.

Variable specifies the variable to contain the conversation number.

Initiates a DDE conversation between the emulator (the client) and a specified application (the server). Both the service and topic names must be supported by the server application. If more than one DDE server application responds to DDE CONNECT, a conversation is initiated only with the first server responding.

The resulting conversation number (a number from 1-10) is stored in the specified variable. This number is used to specify this conversation in other DDE client commands. A conversation is specified by a service name and a topic. Use DDE TOPICS command to display a list of available DDE servers and topics.

Example: **DDE CONNECT “EXCEL” “DATA.XLS” CONV**

Initiates a conversation with Excel, with a topic of DATA.XLS. Places the resulting conversation number in the variable CONV.

Related topics: DDE DISCONNECT

DDE DISCONNECT

DDE DISCONNECT variable

Where: **Variable** indicates the conversation number of the conversation to disconnect. This should be the same number that was returned from the DDE CONNECT command.

Disconnects the specified DDE conversation. Any DDE advise-links associated with the conversation are removed.

Example: **DDE DISCONNECT ‘CONV’**

Terminates the conversation associated with the conversation number CONV.

Related topics: DDE DISCONNECTALL

DDE DISCONNECTALL

DDE DISCONNECTALL

Disconnects all DDE conversations initiated by the DDE CONNECT command. Any DDE advise-links associated with the conversations are removed.

Related topics: DDE DISCONNECT

DDE EXECUTE

DDE EXECUTE variable “command string”

Where: **Variable** is the conversation number returned by an earlier DDE CONNECT command.
“Command string” contains the command to execute.

This command sends the specified command string to the server to be executed.

Example: **DDE EXECUTE ‘CONV’ “@TEST”**

Assumes that CONV refers to a conversation with another copy of the emulator as the DDE server using the topic ECL. The command sent to the server runs the command file TEST.ECF.

DDE POKE

DDE POKE variable “item name” “value”

Where: **Variable** is the conversation number returned by the DDE CONNECT command.
“Item name” is a string expression that specifies the data item to change.
“Value” is a string expression containing the data to send to the server.

Sends “value” to the named item in the server application of the specified conversation. This command sets the server’s item to a specified value.

Example: **DDE POKE ‘CONV’ “WELCOME” “Hello!”**

Assumes that CONV refers to a conversation with another copy of the emulator as the DDE server using the ECL topic. The variable WELCOME in the server the emulator is set to a message string “Hello!”.

DDE REQUEST

DDE REQUEST variable1 “item name” variable2

Where: **Variable1** is the conversation number returned by an earlier DDE CONNECT command.
“Item name” is a string expression that tells the server what data item is being requested.
Variable2 specifies the variable to receive the value of the data item.

Requests the value of the item from the server application, and stores the value of that data item into the specified variable. This value returned for the item may be an empty string if the DDE REQUEST command fails.

Example: **DDE REQUEST ‘CONV’ “WELCOME” RESULT**

Assumes that CONV refers to a conversation with another copy of the emulator as the DDE server using the ECL topic. The DDE_REQUEST command retrieves the contents of the variable WELCOME from the server and places the value in the emulator‘s variable RESULT.

DDE UNADVISE

DDE UNADVISE variable “item name”

Where: **Variable1** is the conversation number returned by an earlier DDE CONNECT command.

“Item name” is a string expression that tells the server what Advise Data Link is to be terminated.

Removes an existing Advise Data Link for the specified item.

Example: **DDE UNADVISE ‘CONV’ “COUNT”**

Assumes that CONV refers to a conversation with another copy of the emulator as the DDE server using the ECL topic, and that an advise-link exists to its variable COUNT. The DDE UNADVISE command removes the Advise-Data Link.

Related topics: DDE ADVISE

DDE TOPICS

DDE TOPICS “service name” “topic name” variable

Where: **“Service name”** is a string expression that corresponds to a DDE server application name. An empty string (“”) can be used as a wildcard to find all DDE server applications.

“Topic name” is a string expression that corresponds to the desired DDE conversation topic. An empty string (“”) can be used as a wildcard to find the DDE conversation topics.

Variable specifies the variable to receive the server/topic list.

Builds a tab-separated list of DDE server application(s) and topic(s) that are currently running. This list only contains the server applications that match the name and name specification parameters. The list is stored into the specified variable as a string, and is empty if a match is not found.

Example 1: **DDE TOPICS “” “” TLIST**

Creates a list of all DDE server applications that are currently running and places this list into the variable TLIST.

Example 2: **DDE TOPICS “” “SYSTEM” TLIST**

Stores a list of all DDE servers that support the System topic into the variable TLIST.

DELAY

DELAY [dd:hh:mm:]ss

Delays the specified amount of time. All of the fields are optional with the exception of seconds. Maximum value is 99:23:59:59.

DELAY is intended for command file use. DELAY does not prevent the emulator from accepting emulator commands sent from the host computer using a DCS private control sequence.

Valid options:

/NODISPLAY

Data received from the host is not displayed on the screen during the delay period.

/NOMESSAGE

Disables display of the delay message.

Example 1: **DELAY 5**

Delays command file execution for five seconds.

Example 2:
```
@LOGIN
DELAY/NODISPLAY 5
WRITE HOST “ACCOUNTING”
EXIT
```

Automatically logs a user in, prevents all login messages from displaying on the screen and starts an accounting application on the host.

Example 3:
```
LOG/OPEN SYSLOG.LOG
DELAY/NOMESS 23:59
LOG/CLOSE SYSLOG.LOG
```

Creates the SYSLOG.LOG file on the PC. Captures information for one day and closes the file.

DELETE SYMBOL

DELETE SYMBOL symbol-name

Deletes a symbol name from the local and/or global symbol table. The symbol name is required. Wildcarding is supported. The default is /LOCAL.

Valid options:

/GLOBAL

Deletes the symbol name from the global symbol table.

/LOCAL

Deletes the symbol name from the local symbol table.

Example 1: **DELETE SYMBOL *A**

Deletes all the local symbols that end with “A”.

Example 2: **DELETE SYMBOL/GLOBAL VARI??**

Deletes all the six letter global symbols that start with “VARI”.

DISPLAY

DISPLAY [[row,column]] [string-expression]

Where: **string-expression** is a quoted string, lexical, symbol, or combination of the above joined by plus signs (+) (i.e., “string” + symbol).

Displays single or multiple lines of text to the screen. DISPLAY can process terminal escape sequences, lexicals, and symbols as part of the string expression. The terminal escape sequence is processed by the selected terminal type when displaying the emulation window.

An initial cursor position can be optionally specified in brackets [ ] immediately following the DISPLAY command. If specified, the cursor moves to the position indicated before the string displays. Specifying a cursor position of 0 for the row or column positions the cursor at the current row or column position.

By default, data is output to the emulation window. Data can be displayed on the status line or to a dialog box by by using the /STATUS and /DIALOG options. DISPLAY will send a carriage return and line feed unless the /NOCR option is used.

Note: Using cursor positioning while outputting data to the status line produces unusual results.

Valid options:

/DIALOG

Displays the text defined by the string-expression in a dialog box.

/NOCR

Do not send a carriage return and line feed.

/STATUS

Displays the text defined by the string-expression on the Status Line.

Example 1: **DISPLAY “Hello there”**

Displays **Hello there** at the current cursor position.

Example 2: **DISPLAY [0,40] “Hello there”**

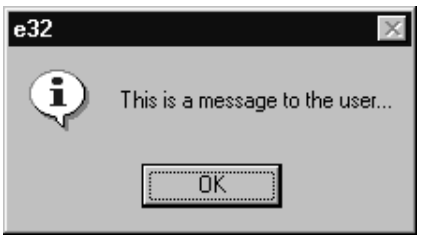
Displays **Hello there** at the current row, column 40 on the screen.

Example 3: **DISPLAY** or **DISPLAY “”**

Outputs a carriage return and line feed at the current cursor position.

Example 4: **DISPLAY /DIALOG “This is a message to the user.”**

This example would yield the following dialog box.



Note: The D$BLOCK lexical is not supported with the /DIALOG option.

Example 5:
```
! ... Additional commands
DISPLAY/NOCR “<CSI>0;0|”                ! enable user def. status line
DISPLAY/NOCR “<CSI>0;2|”                ! erase status line
                                        ! status line message
DISPLAY/NOCR “<CSI>0;3;20| Press ABORT to exit”
! ... Additional commands
```

This example uses DEC terminal escape sequences.

Example 6: **DISPLAY/STATUS “<ESC>[?3h” + “132 columns”**

Sets the screen to 132 column mode, and displays “132 columns” on the status line.

Related topics: INQUIRE, Special Features

DOS

DOS [DOS command string]

Executes the DOS command string and returns to the emulator.

If a DOS command string is unspecified, the DOS shell window appears. Any valid DOS command can be entered in the DOS shell window. To exit from DOS, type EXIT followed by a carriage return.

If a DOS command string is specified, the emulator executes the DOS command and holds the DOS screen. Pressing any key returns closes the DOS shell window and returns to emulation mode.

When the DOS command is issued by the host computer or from a command file, the emulator automatically returns to emulation mode without waiting for keyboard input.

Symbols can be used to assign DOS command strings to a more convenient form. For example, DIR :== “DOS DIR” creates an emulator command that lists DOS directories.

Valid options:

/NOWAIT

When specified interactively, the DOS screen is not held until a key is pressed. The DOS command executes and returns to the emulator without pausing. It has no effect when used in a command file.

Example 1: **DOS TYPE READ.TXT**

Executes the DOS command TYPE and displays the file READ.TXT in a DOS window.

Example 2: **TYPE :== "DOS TYPE"**
TYPE READ.TXT

Creates an ECL command TYPE, then displays the DOS file READ.TXT in a DOS window.

Example 3: **DOS/NOWAIT DEL TEST.LOG**

Switches to a DOS window, deletes the TEST.LOG file, and returns to emulation mode.

DROPDTR

DROPDTR milliseconds

Drops the DTR (Data Terminal Ready) and RTS (Request to Send) lines for the number of milliseconds specified. If milliseconds is zero or missing, DTR and RTS will be dropped permanently.

EMULATE

EMULATE [match-string-expression]

Puts the emulator into emulation mode from a command file. If emulation mode has been entered from a command file, pressing **EXIT** returns to the calling command file rather than to Windows.

The EMULATE command can be used with the ON DISCONNECT command to enter emulation mode and return to a command file when the connection is lost or the user logs out.

Valid options:

/CASE

Force case sensitivity for the return string comparison. /CASE is invalid when used without the /RETURN_STRING option.

/LABEL=label

Resume execution of the command file at the specified label. /LABEL is invalid when used without the /RETURN_STRING option.

/RETURN_STRING = [match-string-expression]

Allows a command file to enter emulation mode and returns control to the command file when a specific string occurs. This option is an alternate form of [match-string-expression] argument. If both strings are used, the first string following the EMULATE command takes precedence.

Allows a command file to enter emulation mode and return control to the command file when a specific string occurs. Execution of the command file resumes at the line immediately following the EMULATE command unless the /LABEL option is used.

Example:
```
50:     SET DISCONNECT
        ON DISCONNECT THEN GOTO 100
        EMULATE
        EXIT/EM                 !USER LOGGED OUT
                                !CONNECTION LOST
100:    DISPLAY “ATTEMPTING TO RECONNECT”
        @RECONNECT
        IF $STATUS GOTO 50
        DISPLAY “UNABLE TO RECONNECT”
        EXIT/EM
```

Monitors connect status. If the connection is lost the command file tries to reconnect.

END INTERACTIVE

ENDINTERACTIVE (no arguments)

Terminates interactive mode. This command is not used in command files.

Related topics: INTERACTIVE

ERASE SCREEN

ERASE SCREEN (no arguments)

Erases the screen.

Example:
```
ERASE SCREEN
DISPLAY [10,20] “1.   Connect Session 1"
DISPLAY [11,20] ”2.   Connect Session 2"
DISPLAY [13,20] “3.   Exit emulator”
INQUIRE [14,20] “Enter menu option number: ”
...
```

Erases the screen before displaying a menu and sends the cursor to Row 1, Column 1.

Related topics: CLS

EXIT

EXIT [specific-error]

Where: **specific-error** is an error code, quoted mnemonic identifier, or symbol. (i.e., EXIT $STATUS)

Terminates processing of the current command file.

EXIT's behavior differs, depending on the mode of usage (interactive or command file mode). If used in interactive mode without an error parameter, the emulator exits to Windows. If used with a parameter, the message associated with the error parameter displays, and no other action is taken.

If used within a command file without a parameter, EXIT passes the error status to the calling routine. If error checking is enabled and an error parameter is provided, EXIT prints the associated error message.

EXIT passes the status and severity codes of the error to the symbols $STATUS and $SEVERITY. It also saves the mnemonic for the error in the symbol $STATUSID and the full error message in F$MESSAGE. If the error message has displayed, bit 15 of the $STATUS symbol is set to 1.

If EXIT is issued from a command file while in emulate mode, emulate mode is exited and the next command is executed.

Valid options:

/EM

Exit the emulator and return to Windows with the corresponding $STATUS code passed to ERRORLEVEL. An exit to Windows leaves the modem control signals active. Refer to the *DOS ERRORLEVEL* topic for more information.

Example 1:
```
EXIT
```
Exits the emulator and returns to Windows.

Example 2:
```
LOG FILELIST          ! Create FILELIST.LOG file
DELAY 1:00: 00        ! Delay 1 hour
LOG/CLOSE             ! Close log file
EXIT                  ! Exit to emulation mode
```
Opens FILELIST.LOG, captures host information for 1 hour, closes the log file, and exits.

Example 3:
```
@SET HOST /DISCONNECT
DELAY/NOMESSAGE 2
EXIT/EM
```
Disconnects from the host, hides all messages and exits the emulator.

Related topics: ON, SET ABORT, SET DEVICE_ERROR, SET DISCONNECT, SET ON, Error Facility, SET MESSAGE

FILE

FILE operation protocol filename

Where: **Operation** is SEND or RECEIVE.
Protocol is one of the available protocols: ASCII, KERMIT, XMODEM, YMODEM, or ZMODEM.
Filename is the name of the file to transfer.

Performs a file transfer using the specified protocol.

Valid options:

/RENAME

Used with RECEIVE to rename incoming files if they would replace an existing file.

FLUSH

FLUSH (no arguments)

Empties the emulator receive buffer to the screen. Used to insure that all data received from the host has been removed from the receive buffer and displayed on the screen.

Related topics: WAIT

GOSUB

GOSUB label_name

Transfers execution to a subroutine label located within the command file. Use the RETURN command to exit the subroutine and resume execution in the calling routine. The calling routine continues at the line following the GOSUB command. (Usable in command procedures only.)

Related topics**:** ON, IF, Labels, RETURN

GOTO

GOTO label-name

Transfers program control to the statement following the specified label. (Used in command procedures only.)

Related topics: ON, IF, Labels

HELP

HELP [keyword]

Displays useful information about emulator operation, key assignments, features, and commands. Specifying HELP without a keyword displays ***Help - Index***.

IF (CONDITIONAL)

IF condition THEN statement

Tests the value of an expression and executes the statement following the THEN keyword if the test is TRUE. If FALSE, THEN is ignored, and execution continues with the next command line.

The expression is true if the result:

1) Has an odd integer value between 2147483647 and -2147483648.
2) Has a character string value that begins with any of the letters Y, y, T, or t.
3) Has an odd numeric string value between “2147483647" and ”-2147483648".

The expression is false if the result:

1) Has an even integer value between 2147483647 and -2147483648.
2) Has a character string value that begins with any letter except Y, y, T, t.
3) Has an even numeric string value between “2147483647" and ”-2147483648".

Rules:

1) Symbols used in IF condition expressions are automatically substituted.
2) String comparison operators end in the letter **S** (.EQS., .LES., .GTS., etc.). Integer comparison operators do not end in the letter **S** (.EQ., .LE., .GT., etc.).
3) String comparisons are case sensitive. Therefore, CASE and case are considered unequal. To inhibit case sensitivity, create the symbol using an implied literal string (:). The string converts to all caps, and can then be compared. (e.g., in the assignment upper := case, the value of upper is converted to CASE.)

Example 1:
```
COUNT = 0
LOOP: COUNT = COUNT + 1
...
IF COUNT .LE. 10 THEN GOTO LOOP
```
This routine loops 10 times.

Example 2:
```
INQUIRE ANS “Want to continue [Y/N] (D:N)”
IF .NOT. ANS THEN EXIT
```
This routine exits unless ANS = Y.

Related topics: Symbols, Lexicals, Error Facility

INQUIRE

INQUIRE[[row,column]] symbol-name [prompt-string]

Where: **prompt-string** is a quoted string, lexical, symbol, or combination of the above joined by plus signs (+) (i.e., prompt-string = “string”+symbol).

Outputs a prompt string and waits for input. The input string is stored in the symbol-name specified. By default, the symbol-name is a local symbol. To make the symbol global, use the /GLOBAL qualifier.

Like the DISPLAY command, the INQUIRE command can process terminal escape sequences, lexicals, and symbols in the prompt string.

An initial cursor position can be specified in brackets [ ] immediately following the command. If specified, the cursor moves to the position indicated before the prompt string displays. Specifying a position of 0 for the row or column positions the cursor at the current row or column on the screen.

By default, INQUIRE uses the screen. However, INQUIRE uses the status line when the /STATUS option is used.

Note: Using cursor positioning while outputting data to the status line or dialog box can produce unusual results and should be avoided.

INQUIRE will not send a carriage return or line feed unless it is placed within the prompt string or the /CR option is used for a single line of text.

Valid options:

/CASE

By default, INQUIRE/KEY is not case sensitive. It does not return the S^ indicator with the key names for alphanumeric keys. Specifying /CASE returns the S^ indicator with uppercase alphanumeric keys. /CASE is only meaningful when used with the /KEY option.

/CR

Send a carriage return at the end of the prompt string.

/DIALOG symbol-name [prompt-string]

Prompts the user for input from a dialog box rather than from the text emulation window. The user supplies a symbol name and a prompt string. The dialog box displays the prompt string and an edit field in which the user can type the symbol value.

/GLOBAL

The symbol name is defined as global.

/KEY

Reads a single keystroke and returns its ASCII key name. The name returned is the same name displayed when the key is pressed in Scan mode. Key remapping is disabled when /KEY is used. /KEY is useful for obtaining a single PC keystroke, such as an arrow key.

/LOCAL

The symbol name is defined as local. This is the default INQUIRE condition.

/MAX=count

Sets the maximum character count for an INQUIRE input line. If the input data exceeds the max count, the extra characters are ignored. The input line is not terminated until a carriage return is entered unless the /TERMINATE option is specified.

/NOECHO

Input data is not echoed to the screen.

/STATUS

Send the prompt string to the status line.

/TERMINATE

Used with the /MAX option to allow an input line to be terminated when the maximum character count is reached. When /TERMINATE is specified, the input line terminates on a carriage return or when the maximum number of characters has been entered. /TERMINATE has no meaning when used without the /MAX option.

Example 1:

```
INQUIRE NUMBER “Enter modem phone number to dial: ”
WRITE HOST “ATDT’’NUMBER’”
WAIT/TIME_OUT=30/ERROR=LATER “CONNECT 2400"
@LOGIN
EXIT
LATER:
  DISPLAY ”There is no modem connection, try later."
  EXIT
```

Requests the phone number from the user. The modem is then dialed. If there is a connection, the user is automatically logged in. If the TIME_OUT criteria is met, then an informational message is displayed and the command file is exited.

Example 2:

```
TIME_STR="Enter Time:"
INQUIRE/GLOBAL [5,0] TIME TIME_STR
```

Positions cursor at the 5th line and current column and displays the prompt “Enter Time:”. The user input string is stored in the global symbol TIME.

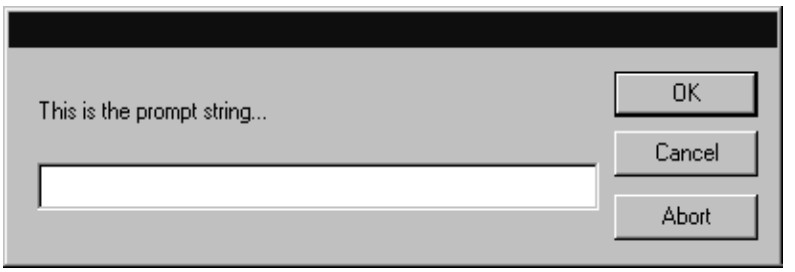
Example 3:

```
50:   INQUIRE/KEY KEYSTROKE “<CR><LF>Enter Up Arrow Key”
      IF KEYSTROKE="UP" THEN GOTO 100
      GOTO50
100:  DISPLAY “<CR><LF>You just pressed the Up Arrow Key”
```

Prompts the user to press the Up Arrow key. The name of the key pressed is stored in KEYSTROKE. A message is displayed once the correct key is pressed. Otherwise, it loops to the beginning for another key press.

Example 4: **INQUIRE /DIALOG THEVAR “This is the prompt string”**

This example would yield the following dialog box.



Note: The D$BLOCK lexical is not supported with the /DIALOG option.

Example 5:
```
WAIT/TIME_OUT=30 “Username:”
WRITE HOST “SMITH”
WAIT/TIME_OUT=30 “Password:”
INQUIRE/LOCAL/NOECHO PASS “Enter your password: ”
WRITE HOST “’’PASS’”
PASS = “”
EXIT
```

Starts the login process for SMITH, then prompts the user for the password. Stores user entry in PASS and sends it to the host. Exits to emulation mode. By defining PASS as a local symbol, it is removed when the exit occurs.

Example 6: **INQUIRE/GLOBAL/NOECHO PASSWD “Password: ”**

Displays the prompt string “Password:” on the screen. The input string is stored in the global symbol PASSWD. The input string is not echoed when it is entered.

Example 7: **INQUIRE/STATUS TIME “World time: ”**

Outputs **World time:** to the status line and stores the input string in the local symbol TIME.

Related topics: DISPLAY, Display functions, Lexicals, Symbols

INTERACTIVE

INTERACTIVE (no arguments)

Sets interactive command mode. Interactive mode is used to enter consecutive commands without clicking ***Execute - Command Line*** each time. This command has little meaning in command files.

To cancel interactive mode, enter the ENDINTERACTIVE command.

Related topics: ENDINTERACTIVE

KERMIT

KERMIT [kermit command string]

Enters Kermit mode. If a command string is not specified, the `KERMIT>` prompt appears. If a string is specified, the emulator enters Kermit mode, issues the command and returns to emulation mode.

Example:
```
WRITE HOST "KERMIT"
WRITE HOST "SET FILE TYPE BINARY"
WRITE HOST "SERVER"
KERMIT SEND/END TEST.EXE
WRITE HOST
WRITE HOST "EXIT"
```

Automatically sets the host Kermit for a binary file transfer, uploads the PC file, TEST.EXE, and exits the host Kermit mode.

KERMIT BYE

KERMIT BYE (no arguments)

Tells the remote server to logout. The emulator terminates the host session and exits.

KERMIT CONNECT

KERMIT CONNECT (no arguments)

Exits from emulator Kermit mode and returns to host Kermit mode. Does not send any commands to the host Kermit. (Equivalent to pressing **Kermit** while in host Kermit mode.)

KERMIT DOS

KERMIT DOS [DOS cmd string]

Displays an DOS Shell window. If a DOS command string is not specified, an active DOS shell window appears. Any valid DOS command can be entered in the DOS shell window. To return to emulation mode, type EXIT.

If a DOS command is specified, an active DOS Shell window displays the result of the command. Click on the X in the upper right corner of the window and select close to return to emulation mode.

When a DOS command is issued by the host computer or from a command file, the emulator automatically returns without waiting for keyboard input.

KERMIT END

KERMIT END (no arguments)

Tells the host server to exit and returns to emulation mode. The host returns to the KERMIT> prompt or to the system prompt. The action taken depends on the host Kermit implementation.

KERMIT EXIT

KERMIT EXIT (no arguments)

Exits the emulator. EXIT does not send any command to the host Kermit.

KERMIT FINISH

KERMIT FINISH (no arguments)

Tells the host server to exit. The Emulator remains in Kermit mode. The host returns to the KERMIT> prompt or to the system prompt. The action taken depends on the host Kermit implementation.

KERMIT GET

KERMIT GET [switches] source file [destination file]

Sends a GET command to the server. This causes the server to send the file or files matching the source file specification to the PC.

The destination file specification is optional. If supplied, the source file is renamed to the destination filename on the PC. The destination filename can include a path specification.

Multiple files can be received with one GET command by separating the filenames with commas or by using wildcards.

Valid options:

/END
Terminates host server mode and returns to emulation mode after successful file transfer.

/EOF
Stores a DOS EOF (Ctrl Z) as the last character of the files transferred.

/LOGOUT
Terminates the host session and returns to emulation mode after successful file transfer.

Examples: **GET *.DAT \DATA*.***
GET *.DAT \DATA
GET *.DAT \DATA

Transfers all .DAT files from the host to the \DATA subdirectory.

KERMIT LOGOUT

KERMIT LOGOUT (no arguments)

Same as the BYE command.

KERMIT RECEIVE

KERMIT RECEIVE [switches] [d-file]

Receives files from a host running Kermit in non-server mode. Before a RECEIVE command can be issued, the SEND command must be given to the host Kermit.

Wildcarding is supported. When using wildcards in the host SEND command, do not specify a destination filename.

A destination filename is only required if you wish to rename the host file being sent.

Valid options:

/EOF

Store a DOS EOF (Ctrl Z) as the last character of the file.

Examples: **RECEIVE**

Transfers all files sent to the default file transfer directory as specified in ***Setup - General - Directories***.

RECEIVE \DATA

Transfers all files sent to the PC's \DATA subdirectory. When using the RECEIVE command, you must include the trailing backslash (\) on the path specification.

KERMIT SEND

KERMIT SEND [switches] source file [destination file]

Sends the source files specified to the host Kermit program. Works with server or non-server Kermit programs. If the host Kermit program is not in server mode, the RECEIVE command must be issued to the host Kermit program before issuing the SEND command.

The file sent can be renamed or sent to a particular directory on the host computer by supplying the optional destination field. Wildcarding is supported.

If host directory strings are used in destination file specification, the host Kermit program should not translate filenames received from the PC. To disable filename translation, issue the following command to the host Kermit:

SET FILE NAMING LITERAL

Note: This is the VAX/VMS syntax for the command. Its syntax may vary on other systems or it may not be supported.

Valid options:

/END

Terminates host server mode and returns to emulation mode after successful file transfer.

/LOGOUT

Terminates the host session and returns to emulation mode after successful file transfer.

/NOEOF

Do not send an EOF (Ctrl Z) character to the host even if Ctrl Z is in the DOS file.

Example: **SEND *.DAT [TEST]**

Transfers all .DAT files to the [TEST] subdirectory on a VMS host.

LOG

LOG filename

Opens an emulator log file. A log file captures all data received from the host. If the file exists and /OVERWRITE or /APPEND is not specified, an error results. The default is /OPEN. The default extension is .LOG.

Valid options:

/APPEND

Open a log file and append the log data to the end of file. If no file exists, one is created.

/CLOSE

Close the previously opened log file. The filename is not required.

/OPEN

Create a log file.

/OVERWRITE

Open a log file and overwrite any old copies. If no file exists, one is created.

/PROMPT

Displays the interactive log file prompt. If logging is already enabled, LOG/PROMPT closes the log file and disables logging. If /PROMPT is used, any other option on the command line is ignored.

Example 1: **LOG TEST**

Creates log file TEST.LOG. If TEST.LOG already exists, an error occurs.

Example 2:
```
INQUIRE TIME "ENTER CURRENT DATE AND TIME: "
OPEN/WRITE ERRORS ERRMESS.LOG
WRITE ERRORS TIME
CLOSE ERRORS
LOG/APPEND ERRMESS.LOG
WRITE HOST "@BUILD"
WAIT "$"
LOG/CLOSE ERRMESS.LOG
```

Creates a log file with a date and time stamp which captures error messages generated from running a VMS COM file.

Example 3:
```
LOG/CLOSE
```

Closes the log file.

Example 4:
```
LOG/OVER TEST
```

Opens TEST.LOG and overwrites any old copies.

Example 5:
```
WRITE HOST "MAIL"
WRITE HOST "READ"
LOG MAIL
WRITE HOST "EXIT"
```

Captures a host mail message into a MAIL.LOG file.

ON ABORT

ON ABORT THEN statement

Defines the course of action when a command file is aborted. The specified action is taken only if the command processor is enabled for abort error checking. Abort error checking is enabled (SET ABORT) by default.

An ON ABORT action remains in effect until one of the following occurs:

- The command procedure exits, which resets to the ON ABORT condition previously specified.
- Another ON ABORT command is executed.
- The procedure executes the SET NOABORT command.

The default error condition is ON ABORT THEN STOP. If an ABORT action is specified, it overrides actions specified for previous levels, and sets the default action for any following sublevels to EXIT. The error codes and mnemonic identifier are stored in the global symbols $STATUS, $SEVERITY, and $STATUSID, even if error checking is disabled (SET NOABORT).

Related topics: SET ABORT

ON DEVICE_ERROR

ON DEVICE_ERROR THEN statement

Defines the course of action when an error occurs from a peripheral device, such as a printer or a plotter. The action is taken only if device error checking is enabled (SET DEVICE_ERROR). By default, device error checking is disabled (SET NODEVICE_ERROR).

An ON DEVICE_ERROR action remains in effect until one of the following occurs:

- The command procedure exits, which restores the previous ON DEVICE_ERROR condition.
- Another ON DEVICE_ERROR command is executed.
- The procedure executes the SET NODEVICE_ERROR command.

The default error condition is ON DEVICE_ERROR THEN STOP. If a DEVICE_ERROR action is specified, it overrides the actions specified for previous levels and sets the default action for any following sublevels to EXIT. When errors occur, the error codes and mnemonic identifier are stored in the global symbols $STATUS, $SEVERITY, and $STATUSID, even if error checking is disabled (SET NODEVICE_ERROR).

Related topics: SET DEVICE_ERROR

ON DISCONNECT

ON DISCONNECT THEN statement

Defines the course of action when the communications connection is lost. The action is taken when the disconnect occurs.

When using an RS232 Serial connection, the Carrier Detect signal is monitored to determine the state of the connection. However, if Modem Control is disabled in the Port Setup dialog box, the state of the connection is not monitored.

When running over a network, the state of the network virtual circuit is monitored.

The specified action is taken only if disconnect error checking is enabled (SET DISCONNECT). By default, disconnect error checking is disabled (SET NODISCONNECT).

An ON DISCONNECT action remains in effect until one of the following occurs:

- The command procedure exits, which restores the previous ON DISCONNECT condition.
- Another ON DISCONNECT command is executed.
- The procedure executes the SET NODISCONNECT command.

The default error condition is ON DISCONNECT THEN STOP. If a DISCONNECT action is specified, it overrides actions specified for previous levels, and sets the default action for any following sublevels to EXIT. When errors occur, the error codes and mnemonic identifier are stored in the global symbols $STATUS, $SEVERITY, and $STATUSID, even if error checking is disabled (SET NODISCONNECT).

Related topics: SET DISCONNECT

ON (ERROR_SEVERITY)

ON error_severity THEN statement

Defines the course of action taken when an error occurs that is equal to or greater in severity than the specified error.

The default error condition is ON ERROR THEN EXIT. This condition tells the command process to CONTINUE when a WARNING error occurs, and execute an EXIT command when an ERROR or SEVERE_ERROR condition occurs. The action is taken only if error checking is enabled (SET ON). Error checking is enabled by default.

These keywords are listed in order of severity and summarize how the command controls error handling:

WARNING	The action is performed if a WARNING, ERROR, or SEVERE_ERROR occurs.
ERROR	The action is performed if an ERROR, or SEVERE_ERROR occurs. Does not affect the handling of warning errors.
SEVERE_ERROR	The action is performed if a SEVERE_ERROR occurs. Does not affect the handling of warning and error conditions.

An ON command action is executed only once. After the ON command action is taken, the ON action is reset to the default (ON ERROR THEN EXIT).

An ON command action can be specified for each active command level. The ON command action applies only within the command procedure in which it is executed. Upon exiting a command procedure, the prior ON error conditions are re-established to their previous settings. The error codes and mnemonic identifier are stored in the global symbols $STATUS, $SEVERITY, and $STATUSID, even if error checking is disabled (SET NOON).

Note: If the command file contains a GOTO command to a non-existent label, an EXIT command executes, regardless of the current ON ERROR assignment.

Related topics: SET ON

OPEN

OPEN logical-name[:] file-specification

Where: **logical-name** is the name used by other commands to reference the open file.

file-specification is the file to open and can include a full path name if desired. The default file extension is .DAT.

Opens a file for read, write, or append operations and assigns a logical name to the file. This command must precede a READ or WRITE command for file access. The file stays open until the CLOSE command is executed or an application exit occurs. If the command file terminates before the opened file is closed, the file remains open.

The same file may be referenced by several open statements. However, each open statement must use a different logical name.

Note: The logical name HOST does not have to be opened before reading or writing.

Valid options:

/APPEND

Opens an existing file for write, starting at the end of the file. If the file does not exist, it is created.

If the /READ option is included with /APPEND, the file must already exist. If the file does not exist, an error occurs.

/ERROR=label

Continues the process at the label if an error occurs.

/READ

Opens an existing file for read only and sets the file data pointer to the beginning of the file. This is the default for the OPEN command.

/WRITE

Creates a new file for write only. If the file already exists, it is overwritten when the first WRITE occurs.

If the /READ option is included with /WRITE, an existing file is opened at the beginning of the file. The file must already exist, otherwise an error occurs.

If the /APPEND option is used with /WRITE, the /WRITE option is ignored.

Example 1: **OPEN FILE2 DATA.TXT**

Assigns DATA.TXT to the logical FILE2, and opens the file named DATA.TXT for reading. An error results if the file does not exist.

Example 2:
```
TOP: INQUIRE/STATUS FILE "Enter the data file name:"
  OPEN/READ/APPEND/ERROR=ERR DATA 'FILE'
  @PROCEDURE
  CLOSE DATA
  DISPLAY "'FILE' has been updated."
  EXIT
ERR:
  DISPLAY "'FILE' does not exist"
  GOTO TOP
```

Checks for the filename entered by the user. If the file exists, PROCEDURE.ECF is run. If the file does not exist, an error message displays and the command file runs again.

Example 3: **OPEN/WRITE FILE1 C:\EM320\TEST.DAT**

Assigns TEST.DAT to the logical FILE1, and creates a file named TEST.DAT for writing.

Related topics: CLOSE, READ, WRITE

PRINT CLOSE

PRINT CLOSE (no arguments)

Closes the open printer, flushes the page and sends the document to the spooler to be printed.

PRINT EJECT

PRINT EJECT (no arguments)

Ejects a page on the printer.

PRINT ON/OFF

PRINT on/off

Turns auto print mode on or off. In auto print mode, every line sent to the screen is also sent to the printer.

Valid options:

/CONTROLLER on/off

Sets printer controller mode in which data passes directly to the printer without displaying on the screen. Use the /CONTROLLER options to print lines longer than 132 columns to pass control characters.

PRINT SCREEN

PRINT SCREEN (no arguments)

Prints the screen (text screen).

PRINT SCROLLBACK

PRINT SCROLLBACK

Prints the text in scrollback memory plus the current screen.

PRINT SELECTED

PRINT SELECTED

Prints the current selection.

QUIT

QUIT [specific-error]

Where: **specific-error** is a quoted mnemonic identifier, error code or a symbol (e.g., EXIT $STATUS).

Works exactly like EXIT except that it drops the modem control signals. See EXIT for a description.

Valid options:

/EM

Quit the emulator and return to DOS with the corresponding $STATUS code passed to ERRORLEVEL. See the *DOS ERRORLEVEL* topic for more information.

READ

READ logical-name[:] symbol-name

Where: **logical-name** is the logical name assigned by an OPEN command or the HOST logical.

Reads an ASCII record from the logical into the specified symbol.

If the READ command references a DOS file, the file is read a record at a time. After each read, the file data pointer is positioned to the start of the next record. The maximum record size is 255 characters. Records are terminated by carriage returns. READ is not intended for use with binary files.

Valid options:

/END_OF_FILE=label

Control transfers to the label when the end of the file is detected. If /END_OF_FILE is not used, and the EOF character is encountered, the process continues at the /ERROR label specified. If neither option is specified, and the EOF character is encountered, the current ON condition is taken. Valid only with a DOS file logical.

/ERROR=label

If an error occurs, control is transferred to the label specified. If /ERROR is not used, the current ON condition action is taken.

Related topics: OPEN, WAIT, WRITE

READ HOST

READ HOST

Reads an ASCII record from the currently connected host into the specified symbol.

Valid options:

/ERROR=label

If an error occurs, control is transferred to the label specified. If /ERROR is not used, the current ON condition action is taken.

/NODISPLAY

Does not display data as it is read. Valid only with the HOST logical.

/TIME_OUT=[hh:mm:]ss

Waits for data until the time specified. Valid only with the HOST logical. A timeout error occurs if no data is received from the host within the specified time. /TIME_OUT and /ERROR can be specified simultaneously to redirect command execution.

Related topics: OPEN, WAIT, WRITE

READ SCREEN

READ SCREEN [row,col] Symbol-name

Where: **row** is the row of the screen to read.

col is the column of the screen to start reading. If col is not specified, column 1 is used.

Reads a specific row of text from the screen into the symbol.

Example: **READ SCREEN [1,10] TEXT**

Reads all the text on line 1 of screen, starting at column 10, into the variable text.

REPLAY

REPLAY filename

Replays an emulator log file. The filename can contain a full path specification and has a default extension of .LOG. Refer to the *Log File Replay* topic for more information.

Valid options:

/PROMPT

Displays the log file prompt.

Example:
```
DISPLAY/NOCR "<CSI>0;0|"
DISPLAY/NOCR "<CSI>0;3;20|Press Alt A to end demonstration."
COUNT=0
TOP:
  CLS
  DISPLAY [5,10] "This demo shows application menus."
  REPLAY MENU1.LOG
  DELAY/NOMESS 10
  REPLAY MENU2.LOG
  DELAY/NOMESS 10
  !... additional replay commands
  COUNT = COUNT + 1
  IF COUNT .LT. 10 THEN GOTO TOP
  DISPLAY/NOCR "<CSI>0;1|"
  EXIT
```

Runs a repeating demonstration program of application menu log files. The user-defined status line is used for messages.

Related topics: LOG

RETURN

RETURN (no arguments)

Used to return from a subroutine called by the GOSUB command. Valid only with the GOSUB command.

Related topics: GOSUB

SCAN

SCAN (no arguments)

Enters keyboard scan mode. In scan mode, pressing a key displays its key name. Scan mode is useful for identifying key names.

SEND

SEND filename

Sends an ASCII text file to the host.

Flow control to the host is provided through character delay (SET CDELAY), line delay (SET LDELAY) and use of the turnaround character (SET TURNAROUND).

Valid options:

/ANSWERBACK

Send the answerback message specified in the ***Setup - Terminal*** dialog box to the host. Since the answerback message can be concealed, store your password in the answerback message when automatically sending it to the host in a command file.

Note: SEND/ANSWERBACK cannot be used with any other qualifiers.

/EOF

Sends an End of File marker at the end of the file. Ctrl Z is the default. The SET EOF command can be used to change the EOF character sent. To send an EOF character without sending data from a file, use SEND/EOF without specifying a filename.

/FILTER

Removes control characters.

Note: Filter will not pass $^{C}_{R}$, $^{L}_{F}$, $^{V}_{T}$, $^{H}_{T}$, and $^{E}_{SC}$.

/LINEFEED

Normally, the emulator does not send line feeds that are immediately preceded by a carriage return. If the /LINEFEED option is specified, all line feeds in the file are sent to the host.

/NOMESSAGE

Supresses the defaultmessage: **"Sending <filename>"**.

Related topics: SET CDELAY, SET EOF, SET LDELAY, SET TURNAROUND, WRITE.

SESSION

SESSION [path]session-name[ext}

Where: **session-name** is the name of the session file. By default the session file is retrieved from the emulators session directory. However, an optional path and file extention can be supplied.

Starts an emulator session using the specified session file.

If the session file does not exist, an invalid file specification error, **Status =STS_K_CMD_INVFSPEC**, will be returned.

SET [NO]DDEAUTOINITIALIZE

SET [NO]DDEAUTOINITIALIZE (no arguments)

Sets the DDE auto initialize feature to on or off. When enabled, the emulator automatically enables itself as a DDE server and broadcasts its name to other Windows applications.

SET [NO]DDEAPPENDINSTANCE

SET [NO]DDEAPPENDINSTANCE (no arguments)

Sets the DDE append instance feature to on or off. When enabled, the emulator appends a unique identifier to the end of the server name. This allows the execution of multiple instances of the emulator while still being able to distinguish them as servers.

Example: **SET DDESERVERNAME "ms320"**
SET DDEAPPENDINSTANCE

Sets the DDE server name for an instance and each subsequent instance. New instances of the emulator automatically append a unique identifier if the Append Unique Identifier option is checked in the DDE Setup dialog box.

SET DDECLIENTTIMEOUT

SET DDECLIENTTIMEOUT seconds

Sets the timeout value, in seconds, for the DDE client commands.

SET DDESERVERNAME

SET DDESERVERNAME "Server Name"

Sets the name that the emulator responds to as a DDE server. Clients use this string as the "Service Name" when performing a DDE connect transaction.

This value is linked to the Server Name option in the DDE Setup dialog box.

Example: **SET DDESERVERNAME "ms320"**

Sets the DDE server name to "ms320"

When changing the server name, the emulator disconnects the instance with the old server name, and reconnects with the new server name.

Note: Any active conversations with the old server name are terminated.

SET ABORT

SET [NO]ABORT (no arguments)

Enables or disables error checking of ***Execute - Abort*** during execution of a command procedure.

The SET NOABORT command disables abort error checking and resets the ON ABORT error condition to STOP. The error codes and mnemonic identifier are still updated in the global symbols $STATUS, $SEVERITY, and $STATUSID.

The SET ABORT and SET NOABORT commands apply to all command procedure levels. SET ABORT is the default. (Usable in command procedures only.)

Note: SET NOABORT is not recommended - it can prevent a normal exit from a command procedure. If a command procedure began to loop uncontrollably, it could not be aborted.

Example: **SET NOABORT**
LOG SYSMESS
DELAY 15:00:00
LOG/CLOSE

Logs all data from the host into SYSMESS.LOG on the PC for 15 hours, say 5pm to 8am. If the command file is aborted, the log file remains open.

Related topics: ON ABORT

SET CHARACTER DELAY

SET CDELAY ms

Sets a character delay for the SEND and WRITE commands. The emulator delays the specified number of milliseconds after sending each character. Specify a character delay to slow down the data rate to prevent overrunning the host's terminal buffer. The default value is zero. Maximum value is 255 ms.

Related topics: SEND File, SET LDELAY, SET TURNAROUND

SET DEVICE_ERROR

SET [NO]DEVICE_ERROR (no arguments)

Enables or disables device error checking. A device error can occur from a peripheral device connected to the serial or parallel port, such as a printer or a plotter as a result of an emulator command. Device errors not associated with emulator functions are not monitored.

This command disables error checking and resets the ON DEVICE_ERROR condition to STOP. The error codes and mnemonic identifier are still updated in the global symbols $STATUS, $SEVERITY, and $STATUSID.

The SET DEVICE_ERROR and SET NODEVICE_ERROR commands apply to all command procedure levels. SET NODEVICE_ERROR is the default. (Usable in command procedures only.)

SET DISCONNECT

SET [NO]DISCONNECT (no arguments)

Enables or disables error checking of the communications connection. Disconnect errors can occur when serial or network connections are lost.

This command disables error checking and resets the ON DISCONNECT error condition to STOP. The error codes and mnemonic identifier are still updated in the global symbols $STATUS, $SEVERITY, and $STATUSID.

The SET DISCONNECT and SET NODISCONNECT commands apply to all command procedure levels. SET NODISCONNECT is the default. (Usable in command procedures only.)

Related topics: ON DISCONNECT

SET EOF CHARACTER

SET EOF value

Where: **value** is the decimal value of the ASCII character. Ctrl Z (26) is the default.

Defines the End of File character sent by the /EOF option of the SEND command.

Related topics: SEND File, SET CDELAY, SET LDELAY, SET TURNAROUND

SET HOST

SET HOST [node-name]

Connects to a remote node. The SET HOST command must be used with one of the following:

Valid options:

/DEFAULT_PORT

Connects to the default port selected in the Auto Connect Port dialog box. If the port is set to None or if you are already connected to the default port, an error is returned.

/DISCONNECT

Disconnects from the currently connected port.

/LAST_NODE

Connects to the last successfully connected port. If a previous connection did not exist, an error is returned.

/PROTOCOL= node

Connects to the specified protocol.

Where: **protocol** is SERIAL, MODEM, POLYLAT, WINSOCK, etc,...
node is the network node name.

/PASSWORD=password

Used only with the /PORT option, the /PASSWORD option allows the connect password to be specified.

Example 1: **SET HOST/SERIAL=COM1**
Connects to COM1.

Example 2: **SET HOST/WINSOCK=WILLY**
Connects to the WINSOCK node WILLY.

Example 3: **SET HOST/POLYLAT=MARS**
Connects to the LAT node MARS.

SET KEYMAP

SET KEYMAP name

Where: **name** is the name of a keymap.

Switches the current keymap to the specified keymap.

SET LINE DELAY

SET LDELAY secs

Sets a line delay for the SEND and WRITE commands. Specifies the time for the emulator to wait after sending a line before sending the next line. The default is zero. Maximum value is 255 seconds.

If a line delay and turnaround character is specified, the emulator waits until it receives the turnaround character or the delay expires, whichever occurs first. If SET NOTURNAROUND has been specified, the emulator waits the full delay after each line.

Related topics: SEND File, SET CDELAY, SET TURNAROUND, WRITE

SET MESSAGE

SET [NO]MESSAGE [message_type]

Where: **message_type** is Informational, Warning, Error, or Severe_Error.

SET MESSAGE and SET NOMESSAGE enable and/or disable the display of messages. The message_type determines the category of message affected. All messages below or equal to the message_type specified are affected. If no message_type options are provided, SET NOMESSAGE affects all messages.

Example: **SET NOMESSAGE = WARNING**

Disables informational and warning messages.

SET ON

SET [NO]ON (no arguments)

Enables or disables error checking.

SET NOON disables error checking and error message display. However, the error codes and mnemonic identifier in the global symbols $STATUS, $SEVERITY, and $STATUSID are updated.

The SET ON and SET NOON commands apply only to the current command level. If SET NOON is used in a command procedure that calls a second procedure, the default (SET ON) is used while executing the second command procedure. (Usable in command procedures only.)

Related topics: ON error_severity

SET TERMINAL

SET TERMINAL characteristic

Sets the terminal characteristics.

Valid options:

/APPLICATION_KEYPAD

Specifies that the keypad keys send application control functions. Limited to DEC terminal emulation modes.

/DATA_BITS=bits

Where: **bits** is 7 or 8.

Sets the number of communication data bits. The default is 8 bits with parity = none. Limited to Serial communications.

/DEVICE=terminal

Where: **terminal** is VT320_7, VT320_8, VT220_7, VT220_8, VT100, VT52, SCO-ANSI, or BBS-ANSI.

Selects the terminal to emulate.

/[NO]ECHO

Controls display of input from the keyboard. If ECHO is set, the data transmitted to the host is locally echoed to the screen. If NOECHO is set, the data is not echoed by the emulator. In NOECHO mode the host is expected to echo the data. NOECHO is the default. ECHO should be set on half-duplex systems.

/INSERT

Sets the line editing mode to insert. Limited to DEC terminal emulation modes.

/LIMITED_TRANSMIT

Restricts the transmit speed to between 150 and 180 characters per second. Limited transmit may be necessary for some half-duplex systems. Limited to Serial communications.

/LINES=rows

Where: **rows** is 24 - 48.

Sets the screen height to the desired number of rows.

/LOCAL

Sets the emulator to local mode. In local mode, all characters entered from the keyboard are sent to the screen display processor. Data is not sent to the host and data received from the host is ignored.

/[NO]MODEM_CONTROL

Enables/disables carrier detect monitoring. Modem control should be disabled when using a direct connection. Limited to Serial communications.

/[NO]NEW_LINE

If enabled, generates a line feed whenever a carriage return is entered.

/NUMERIC_KEYPAD

Specifies that the keypad keys send numeric control functions. Limited to DEC terminal emulation modes.

/ONLINE

Allows the emulator to communicate with the host. (Disable with the /LOCAL option.)

/OVERSTRIKE

Sets the line editing mode to overstrike. New characters entered into the line replace the existing characters. /OVERSTRIKE is the system default. Limited to DEC terminal emulation modes.

/PARITY=type

Where: **type** is Odd, Even, Space, Mark or None.

Sets the communications parity. Parity = none and Data Bits = 8 is the recommended default. Limited to Serial communications.

/PORT=com port

Where: **com port** is COM1, COM2, COM3 or COM4.

Selects the communications port. Limited to Serial communications.

/FLOW_CONTROL=type

Where: **type** is XON, RTS, or None.

Selects the communications flow control protocol. Xon/Xoff is the protocol used by DEC and most other host systems. Limited to Serial communications.

/SPEED=baud rate

Where: **baud rate** is 75, 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Selects the communications speed. Limited to Serial communications.

/STOP_BITS=num

Where: **num** is 1 or 2.

Sets the number of stop bits for each data word. One is the recommended setting. Limited to Serial communications.

/UNLIMITED_TRANSMIT

Does not limit the character transmit rate. This is the recommended setting. (The transmit rate can be restricted with the /LIMITED_TRANSMIT option.) Limited to Serial communications.

/[NO]WARNING_BELL

Enables/disables a warning bell for operating errors and receipt of a Ctrl G.

/WIDTH=columns

Where: **columns** is 80 or 132.

Sets the screen width to 80 or 132 columns.

/[NO]WRAP

Controls whether the emulator generates a carriage return and line feed at the end of a line. The end of the line is determined by the /WIDTH option. If /NOWRAP is specified, the characters written at the last column position overwrite each other. /WRAP is the default.

SET TURNAROUND CHARACTER

SET [NO]TURNAROUND value or quoted string

Where: **value** is the decimal value of the ASCII character or a quoted character. Line Feed (10) is the default.

Sets the turnaround character for the SEND command. When a turnaround character is specified, the emulator waits for the turnaround character to be received from the host before sending the next line.

Turnaround characters perform flow synchronization and help prevent overrunning the host's terminal input buffer. If a turnaround character is specified, the SEND operation could hang if a turnaround character is not received. Clicking on ***Execute - Abort*** terminates the operation. If a SET LDELAY is specified with the turnaround character, it is used as the maximum time the emulator waits before sending the next line. (Affects the SEND and WRITE command only.)

Examples: **SET TURNAROUND = 10**
SET TURNAROUND = "<LF>"

Both commands set the turnaround character to a line feed.

Related topics: SEND File, SET CDELAY, SET LDELAY

SET VERIFY

SET [NO]VERIFY (no arguments)

When enabled, displays command lines of a command procedure as they are executed. Also, enables the display of error messages regardless of whether error checking is disabled. The default is SET NOVERIFY.

SHOW SYMBOL

SHOW SYMBOL [symbol-name]

Displays the local and global values for the specified symbol. If no symbol name is given, all the symbols from the local and global symbol table are displayed. Wildcarding is supported; an asterisk (*) may be used for variable length substitution and a question mark (?) for single letter substitution. The default is SHOW SYMBOL /LOCAL/GLOBAL.

Note: Although SHOW SYMBOL displays local and global symbols of the same name, the local value of a symbol will override the global value when referenced in a command procedure.

Symbol values are displayed on the screen regardless of the message location.

Valid options:

/GLOBAL

Displays the value(s) from the global symbol table.

/LOCAL

Displays the value(s) from the local symbol table.

Example 1: **SHOW SYMBOL *A**

Displays all the symbols that end with "A".

Example 2: **SHOW SYMBOL/LOCAL VARI??**

Displays all the six letter local symbols that start with "VARI".

Related topics: DELETE SYMBOL

STOP

STOP (no arguments)

Terminates the execution of all command files.

Related topics: EXIT

WAIT

WAIT [match-string-expression]

Where: **match-string-expression** is a quoted string, lexical, symbol, or combination of the above joined by plus symbols (+) (i.e., "string" + symbol).

Waits for the match string expression to be received from the host. The string must match the host data exactly, but is not case sensitive unless the /CASE option is specified. WAIT is intended for command file use.

If the WAIT command is issued from the host, it does not prevent the emulator from accepting additional host commands while it is waiting for the string.

Valid options:

/CASE

Requires the comparison to be case sensitive.

/ERROR=label

Process continues at the label if an error occurs.

/NODISPLAY

Inhibits the display of data from the host.

/NOMESSAGE

Inhibits the display of the WAIT informational message.

/NOSTRING_DISPLAY

Inhibits the display of the match string.

/TIME_OUT=[hh:mm:]ss

Sets a maximum time period to wait for the host string match. If the string is not received in the allotted time, the process continues with the next command line. Specifying a /TIME_OUT qualifier without a string flushes data received from the host until no data is received for the time specified. The /TIME_OUT option can be used with the /ERROR option.

Related topics: READ, WRITE

WIN

WIN (Windows command string)

Executes the Windows command string in order to launch a Windows application from within the emulator.

Example: **WIN NOTEPAD**

Displays the Windows Notepad.

Symbols can be used to assign Windows command strings to a more convenient form.

Example: **NOTEPAD:=="WIN NOTEPAD"**
NOTEPAD C:\EMULATOR\MODEM.ECF

Creates an emulator command, NOTEPAD, that launches the Notepad editor. The editor then displays the MODEM.ECF file.

WORDPERFECT MODE

WP5 ON/OFF

Enables or disables WordPerfect version 5.0 mode. WP OFF also disables WordPerfect 5.0 mode. In WP mode, the emulator's keyboard assignments are altered to emulate the PC version of WordPerfect.

WRITE

WRITE logical-name[:] [string-expression]

Where: **logical-name** is a file logical assigned by the OPEN command or the HOST logical. HOST is a special predefined local symbol that points to the selected communications port.

string-expression is a quoted string, lexical, symbol, or combination of the above joined by plus signs (+) (i.e., "string" + symbol).

Writes the string expression to the logical name, followed by a carriage return. To suppress the carriage return, use the /NOCR option.

If information is written to a file, the file pointer is positioned after the data written.

Flow control is provided through character delay (SET CDELAY) and line delay (SET LDELAY).

Valid options:

/ERROR=label

Process continues at the label if an error occurs.

/KEY_TOKEN= token

Where: **token** is a valid terminal keyboard token.

Used with the HOST logical, this option sends token value to the host.

/NOCR

No carriage return is sent after the string. A carriage return is sent separately.

/UPDATE

The data previously READ is to be overwritten. Valid only when rewriting the previous record read. The new data string must be the same length as the previous string or an error results. Valid only with a file logical opened with the /READ and /WRITE options.

Example 1: **WRITE HOST**

Sends a carriage return to the host. (Also the same as WRITE HOST “”)

Example 2: **WRITE /KEY_TOKEN=BACKSPACE HOST**

Sends a backspace to the host.

Example 3: **WRITE HOST “SET X:==””ABC”””**

Sends **SET X:=="ABC"** to the host.

Example 4:
```
P1 = XRAY.DAT
WRITE HOST “ TYPE ‘’P1’”
```
Sends **TYPE XRAY.DAT** to the host.

Example 5:
```
READ FILE2 DATA
WRITE/UPDATE FILE2 TEXT
```
Reads the first record from the logical name **FILE2** into the symbol **DATA**, then replaces the data just read with the information in symbol **TEXT**. Both sets of data must be the same length. The DOS file must have been opened using /READ and /WRITE.

Related topics: OPEN, READ, SET CDELAY, SET LDELAY, WAIT

COMMAND FILE PROGRAMMING

OVERVIEW

Command files are DOS text files that contain emulator commands. Command files are useful for automating tasks such as transferring files, logging on, and defining keyboard configurations. However, command files are not limited to the above functions. This chapter is devoted to command language programming while Chapter 1 (Command Language) explains the individual emulator commands.

This chapter covers the following advanced programming features:

- Symbol assignment and substitution
- Full range of lexical functions (Locate, Extract, etc.)
- Logical operations
- IF processing
- Special display lexical functions
- Command file nesting
- Comprehensive error control

2.1 DOCUMENTING COMMAND FILES

It is a good programming practice to use comments to document command procedures. Comments are prefixed with the exclamation point (!). Any data to the right of the exclamation point is ignored. If a literal exclamation point is needed in a command line, it must appear within a quoted string or it is interpreted as the comment character. In this example, the boldfaced type is used to set off the comments.

Example:
```
! This procedure dials a modem number and transfers a text file to the host.
!   Format: SENDTXT input-file [output-file]
!   Where:      P1 is the filename to send, [P2] is the output filename if different
IF P2 .eqs. "" THEN P2 = P1              ! Make sure p2 is defined
ON WARNING THEN EXIT                     ! Set to EXIT if error
DIAL VAX                                 ! Dial phone directory entry VAX
WRITE HOST "COPY TT: ''P2'"              ! Set host to receive data to filename p2
WAIT "<CR><NULL> "                       ! Wait for prompt
ON WARNING THEN GOTO DONE                ! Set to close COPY command if error
SEND 'P1'                                ! Send the file p1
DONE:
SEND EOF                                 ! Close COPY command
EXIT
```

2.2 PASSING PARAMETERS

Up to eight parameters can be passed to a command file. Each parameter must be separated by a space.

Example: **@filename [p1] [p2] ... [p8]**
Or, from the host: **$^{C}S_{I}5|$@filename [p1 p2 ... p8]$^{S}_{T}$**

Commands in the command file utilize the passed parameters by referring to P1 - P8. The value of a passed parameter is recovered by quoting the parameter with the symbol substitution character **'** (single quote). The parameter values are automatically converted to uppercase unless they are enclosed in a set of quotes.

Example: The file SEND.ECF contains the string: **KERMIT SEND 'P1','P2'**

Typing this string at the command prompt: CMD>**@SEND FILE1.DAT FILE2.DAT**

Tells the emulator to issue the command: **KERMIT SEND FILE1.DAT, FILE2.DAT**

2.3 SYMBOLS

A symbol (also known as variables) is a name to which a character string or integer value is assigned. The symbol name must begin with an alphabetic character, an underscore (_) or a dollar sign ($), but may contain other alphanumeric characters. The maximum symbol name length is 31 characters.

Integer values are limited to 16 bits (-32767 to 32767). Strings are a maximum of 255 characters in length and must be quoted (string = “string”) or assigned using the implied string delimiter (string**:=**expression).

Sypmbols can be used for the following purposes:

- Synonyms for emulator commands (foreign commands)
- Variables in expressions or command procedures
- Arguments to commands
- Arguments to command procedures

2.3.1 Symbol Types

There are two types of symbols: Local and Global. Local symbols are available as long as the current command file is executing. Global symbols are permanently defined until deleted or the emulator exits.

The emulator stores symbols in local and global symbol tables. A local symbol table is maintained for each active command level including emulation mode (no command file executing). These tables are deleted as their respective command level is terminated. (Local symbols from all command levels above the current level are available to the current level.) The emulation mode local symbol table is deleted when the emulator is exited.

Note: A new command level is created each time a command file is executed without exiting the current command file (nesting command files).

Global symbols are accessible by all command levels. The emulator maintains only one global symbol table.

Local symbols are assigned using an equal sign (=). Global symbols are assigned with a double equal sign (==).

Example:
```
KER = “KERMIT”          (local)
SS == “SHOW SYMBOL”     (global)
```

2.3.1.1 Permanent Global Symbols

Three permanent global symbols, $STATUS, $SEVERITY, and $STATUSID, are reserved. They hold the error code and error mnemonic from the most recently executed command.

These symbols are useful when nesting command files. When a command file is complete, control returns to the calling command file. The status of the exiting command file is stored in $STATUS, $SEVERITY and $STATUSID for testing by the calling command file. If no error occurs, a status of SUCCESS (1) returns in the symbols.

2.3.2 Assigning Symbol Values

The assignment statement equates a symbol to an expression:

symbol-name =[=] expression

An expression can contain an integer value, a symbol name, a quoted string, a lexical function or a combination of these connected with arithmetic operators. See the Section 2.5.3 (Integer Expressions) for more information.

Example 1:
```
XX == "This is a string"          ! Global String
SHOW SYMBOL XX
XX == "This is a string"
```

Example 2:
```
SUBSTR = F$EXTRACT(5,2,XX)        ! Local String
SHOW SYMBOL SUBSTR
SUBSTR= "is"
```

Example 3:
```
COUNT == 1                        ! Global integer
SHOW SYMBOL COUNT
COUNT == 1 Hex=0001 Octal = 000001
```

Example 4:
```
SS == "show symbol"               ! Global String
TEXT== "This is a test"           ! Global String
SS TEXT
TEXT== "This is a test"
```

2.3.2.1 Implied String Assignments

Use a colon with an equal sign (:= or :==) to specify an implied string assignment. Quotes are not required.

Examples:
```
TEXT:= THIS IS A TEST             (local)
SS:== SHOW SYMBOL                 (global)
```

Leading and trailing tabs and spaces are stripped from implied strings. All other multiple spaces or tabs are reduced to a single space character.

Implied strings are normally converted to all capital letters. Case toggles on and off using a quote sign (").

Example:
```
TEXT:== "This is a "test
SHOW SYMBOL TEXT
TEXT== "This is a TEST"
```

Enclosing the entire string in quotation marks prevents uppercase conversion.

Example:
```
TEXT:= "This is a test"
SHOW SYMBOL TEXT
TEXT== "This is a test"
```

Pair consecutive quotes (“”) together to embed a quotation mark (“) within a string expression.

Example:
```
TEXT:= “This is a ””TEST”” line"
SHOW SYMBOL TEXT
TEXT== “This is a ”TEST" line"
```

Terminate an implied string expression with a carriage return or an exclamation mark (comment character).

Example:
```
TEXT:== This is a ! test
SHOW SYMBOL TEXT
TEXT == “This is a”
```

An exclamation mark can be included within an implied string assignment by quoting the string.

Example:
```
TEXT:== “This is a test!”
SHOW SYMBOL TEXT
TEXT == “This is a test!”
```

2.4 LABELS

Labels are names used to symbolically reference a location within a command file.

Example:
```
LOOP: IF COUNT .EQ. 10 THEN GOTO DONE
COUNT=COUNT+1
GOTO LOOP
DONE: DISPLAY “DONE”
```

Labels are useful for redirecting command file execution (GOTO label). They are also used for marking the beginning of a D$BLOCK text block.

A label is always followed by a colon (:). Any printable ASCII character can be used in a label name. Labels have a maximum length of 32 characters, including the colon.

2.5 EXPRESSION EVALUATION

Expressions evaluate to either string or integer values, depending on the type of value used in the expression and the operator used to modify or compare them. Table 8-1 lists the expression evaluation rules. If “any value” is a string value, it is converted to an integer value before the operation is performed (except string compare).

Table 2-1 Expression Modes

Expression	Result
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
+,-,or NOT any value	Integer
Any value .AND. any value	Integer
Any value .OR. any value	Integer
String + or - string	String
Any value * or / any value	Integer
Any value (string compare) any value	Integer
Any value (arithmetic compare) any value	Integer

2.5.1 String to Integer Conversion

Strings containing numbers are converted to their integer values. For example, the string “64" is converted to 64.

Alphabetical strings are converted to the integer 1 if the string begins with T, t, Y, or y. If the string begins with any other letter, the string is converted to integer 0.

2.5.2 String Expressions

A character string expression is an expression that evaluates to a string value. A character string expression can contain character strings, lexical functions that evaluate to strings, and symbols that evaluate to strings. They can also contain groups of strings connected by operators. Whenever values are connected by one or more operators, all values must be string expressions for the result to remain a string expression.

Examples: **FILENAME= “XRAY.DAT”**

TEXT = “TIME” + “OUT”

COUNT = “TEN”

TOTAL = “THE TOTAL IS ” + COUNT

A String value unrepresented by an alphabetical character is inserted into a string with a pair of angle brackets.

Example: **FF = “<12>”** **!FF = Form Feed**

2.5.3 Integer Expressions

An integer expression is an expression that evaluates to an integer value. An integer expression can contain integers, lexical functions that evaluate to integers, and symbols that evaluate to integers. They can also contain groups of integers or strings connected by arithmetic operators, logical operators, and comparison operators.

Integer values must be specified as decimal numbers unless preceded by a Radix operator. Hexadecimal numbers use %X while Octal numbers are specified using %O.

Examples:
```
COUNT = 10              ! DECIMAL 10
HEX = %XC               ! HEX C
OCTAL = %012            ! OCTAL 12
SUM = 1 + 7 + COUNT
```

2.5.4 Expression Substitution

This feature is useful for debugging when SET VERIFY is in effect. Early evaluation of an expression can be forced by the use of the apostrophe (‘) substitution operator. The expression being evaluated must be enclosed in parenthesis and be preceded by the apostrophe.

Example: **IF ‘(a + b) .eq. ‘(c - d) THEN GOTO END**

The value of **a + b** and **c - d** are evaluated and their values are compared to see if they are equal. If equal, the control continues at label **END**. The apostrophe does not change the final result.

Expression substitution is useful when using SET VERIFY to determine the result of an evaluation.

Formal evaluation of an expression occurs left to right within the parentheses. An error results if the expression is unbalanced causing an unresolvable evaluation. See also, Section 2.12 (Symbol and Lexical Substitution).

Example:
```
SET VERIFY
A = 5
B = ‘A * 2
C = ‘(A + B)
D = ‘((A + B) - C)
IF ‘(A + B) .eq. ‘(C + D) THEN ANS = “TRUE”
IF ‘((A + B) .eq. (C + D)) THEN ANS = “TRUE”
```

Read from a command file, these expressions would evaluate and display to the screen as:

```
A = 5
B = 5 * 2
C = 15
D = 0
IF 15 .eq. 15 THEN ANS = “TRUE”
IF 1 THEN ANS = “TRUE”
```

2.6 OPERATORS IN EXPRESSIONS

Operators connect two or more elements within an expression. Some are mathematical symbols like the plus sign (+). Others specify logical and comparison operations and consist of letters enclosed in a set of periods.

If more than one operator appears in an expression, the operators are executed in order of precedence. The higher the precedence number, the higher the priority of the operator. Operators of equal value are executed from left to right.

Parentheses override the order operators are evaluated. Expressions enclosed in parentheses are evaluated first.

Table 2-2 Operator Precedence

Operator	Precedence	Description
+	7	Unary + (Positive number)
-	7	Unary - (Negative number)
*	6	Multiply
/	6	Divide
+	5	Add two numbers or string concatenation.
-	5	Subtract two numbers or string reduction.
.eqs.	4	String equal test
.nes.	4	String not equal test
.ges.	4	String greater or equal test
.gts	4	String greater than test
.les.	4	String less or equal test
.lts.	4	String less than test
.eq.	4	Equal to
.ne.	4	Not equal to
.ge.	4	Greater or equal to
.gt.	4	Greater than
. le.	4	Less or equal to
.lt.	4	Less than
.not.	3	Logical Negate (1's Compliment)
.and.	2	Logical AND
or.	1	Logical OR

2.6.1 String Operations

String operators are used to concatenate or reduce strings. The + operator is used for concatenation and the - operator is used for reducing a string.

A string concatenation (+) adds two strings together to form a longer string.

A string reduction (-) subtracts two strings by removing the string following the minus sign from the first string. If the second string occurs more than once in the first string, only the first occurrence of the string is removed.

Example 1: **A = “MYFILE” + “.DAT”**
Result: MYFILE.DAT

Example 2: **B = “FILE NAME FILE.DAT” – “FILE ”**
Result: NAME FILE.DAT

Note: When concatenating or reducing strings, both operands must be strings or result in an integer.

2.6.2 Arithmetic Operations

Arithmetic operators are used to perform calculations in integer expressions. The result of an arithmetic operation is an integer. The following operators are valid:

Table 2-3 Arithmetic Operators

Symbol	Operation
+	Add
-	Subtract
/	Divide
*	Multiply
+	Unary plus sign
-	Unary minus sign

If string values are used as operands to arithmetic operations, the strings convert to integers first. See Section 2.5.1 (String to Integer Conversion) for more information.

In arithmetic operations, all non-decimal values (values specified using radix operators) convert to their decimal equivalent.

Examples:

```
A = 5 + 10 / 2                  ! 10
B = 5 * 3 – 4 * 6 / 2           ! 3
C = 5 * (6 – 4) – 8 / (2 – 1)   ! 2
D = –5 + 4                      ! -1
E = 8 + “1"                     ! 9
F = %X1f + %O17 – %D10          ! 36
```

2.6.3 Logical Operations

Logical operators are used to perform logical functions on integers or to create expressions that perform Boolean arithmetic. The result of a logical operation (.NOT., .AND., .OR.) is an integer value.

Examples:
```
A = %X15 .OR. %X12          ! Decimal = 23
A = %X15 .AND. %X12         ! Decimal = 16
.NOT. %X15                  ! Decimal = -22
```

Logical operators can be used in a logical sense as well as arithmetic. An integer has a logical value of true (1) if it is odd (low order bit=1). A character string is true if it begins with Y, y, T, or t. An integer has a logical value of false (0) if it is even (low order bit=0). A string value is false if the first character is not a T, t, Y or y.

Example: **B = %X200 .OR. %X201**

This expression performs a logical OR on two values. The resulting symbol is True and has a value of 513 (odd) or 201 Hex. Of the original operands, 200 Hex is False and 201 Hex is True.

2.6.4 String Comparisons

String comparison operators are used to compare character strings. String comparison results are based on the binary value of the string characters. See Appendix B for a table of ASCII character values. The result of a string comparison is the integer 0 (False) or 1 (True).

The following are the string comparison operators:

Table 2-4 String Comparison Operators

Operator	Definition
.EQS.	String equal to
.GES.	String greater than or equal to
.GTS.	String greater than
.LES.	String less than or equal to
.LTS.	String less than
.NES.	String not equal to

The following rules apply to string comparisons:

- The comparison is on a character by character basis that stops as soon as two characters do not match.
- In comparisons of different length strings, the shorter string is padded on the right with null (00) characters before the operation is performed.
- Lowercase letters have a higher numeric value than their corresponding uppercase letters.

Operands in string comparisons are assumed to be string expressions. If an integer expression is specified as an operand, it is converted to a string before the comparison.

If a character string is not enclosed in quotes, the string is assumed to be a symbol name.

Examples:
```
"ABC" .LTS. "abc"          ! True (1)
"TRUE" .EQS. 1             ! False (0)
"ABC" .GTS. "DEF"          ! False (0)
"CAT" .EQS. "CATS"         ! False (0)

CANDY :== MARS BAR
"MARS BAR" .EQS. CANDY     ! True (1)
```

2.6.5 Arithmetic Comparisons

Arithmetic comparison operators compare integer values. The result of an arithmetic comparison is an integer. If the result is true, the expression result is 1. If the result is false, the expression is evaluated to 0.

The following is a list of the arithmetic comparison operators:

Table 2-5 Arithmetic Comparison Operators

Operator	Definition
.EQ.	Equal to
.GE.	Greater than or equal to
.GT.	Greater than
.LE.	Less than or equal to
.LT.	Less than
.NE.	Not equal to

Operands in arithmetic expressions are assumed to be integer expressions. If a character string is specified as one of the operands, it is converted to an integer before the comparison is performed. If a character string begins with an upper- or lowercase Y or T, it is converted to a 1. If the string begins with any other letter, it is converted to 0. If the string consists of characters that form a valid number, the number is converted to an integer.

2.6.6 Radix Operators

There are three special operators recognized for specifying the radix (number system) for integers. Decimal is the default and %D is not required when specifying decimal values.

Table 2-6 Radix Operators

Operator	Meaning	Example	Decimal Value
%D	Decimal	%D100	100
%X	Hex	%X64	100
%O	Octal	%O144	100

Example:
```
TOTAL = 100 + %X64 + %O144
SHOW SYMBOL TOTAL
TOTAL = 300, HEX = 012C, OCTAL = 000454
```

2.7 SPECIAL CHARACTERS

2.7.1 Input Conversion

ASCII codes that are unspecified by a printable character can be inserted into strings using their numeric value. To specify an ASCII character by its value, enclose its numeric equivalent inside angle brackets < >.

Example: STRING:==**"<7>Attention"**

Inserts a bell into the string by specifying the decimal equivalent for an ASCII bell character.

The most commonly used characters can also be specified by a set of mnemonics.

Table 2-7 Mnemonic Table

Mnemonic	Decimal Value	Mnemonic	Decimal Value	Mnemonic	Decimal Value	Mnemonic	Decimal Value
NULL	00	DLE	16	GS	29	SS2	142
SOH	01	DC1	17	RS	30	SS3	143
STX	02	XON	17	US	31	DCS	144
ETX	03	DC2	18	SP	32	PU1	145
EOT	04	DC3	19	DEL	127	PU2	146
ENQ	05	XOFF	19	IND	132	STS	147
ACK	06	DC4	20	NEL	133	CCH	148
BELL	07	NAK	21	SSA	134	MW	149
BS	08	SYN	22	ESA	135	SPA	150
HT	09	ETB	23	HTS	136	EPA	151
LF	10	CAN	24	HTJ	137	CS	155
VT	11	EM	25	VTS	138	ST	156
FF	12	SUB	26	PLD	139	OSC	157
CR	13	ESC	27	PLU	140	PM	158
SO	14	FS	28	RI	141	APC	159
SI	15						

Example: **STRING:== "<BELL>Attention"**

Unrecognized numeric characters and values greater than 255 are ignored. Radix operators are also supported within the angle brackets.

Conversion of numeric values enclosed in angle brackets is prevented by using a double set of brackets << >>. Using a double set of angle brackets results in a numeric string enclosed in a set of single brackets < >.

Examples:

```
A = “a b c <68>”                    ! “a b c D”
B = “a b c <%X44>”                  ! “a b c D”
C = “a b c <<44>>”                  ! “a b c<44>”
D = “<%X7e>,<<abc>>,<<256>>”        ! “~,<abc>,<256>”
E = “<ESC>[10;20H”                  ! 27"[10;20H"
```

2.7.2 Output Conversion

Non-printable characters and characters specified by enclosing their numeric value in angle brackets < >, are displayed in two ways:

1) Their binary value is sent directly to the screen processor. In this case, the character performs its specific function (e.g., <7> rings the bell) or appears as a character if it is printable (e.g., <%x41> is an A). Commands such as DISPLAY and INQUIRE process data in this manner.

2) The non-printable character or character enclosed in angle brackets displays as a mnemonic or numeric value enclosed in brackets. The output from SHOW SYMBOL and SET VERIFY appears this way.

Example:

```
TEST="<7>This is a test"
SHOW SYMBOL TEST

TEST = “<BELL>This is a test”

DISPLAY TEST

This is a test    (also rings the bell)
```

Non-printable ASCII codes are control characters with numeric values below 32 decimal and ASCII codes with values of 127 to 255. The more frequently used control codes are output as mnemonics instead of decimal values.

Table 2-8 Mnemonic Table Output Conversion

Mnemonic	Decimal Value	Mnemonic	Decimal Value
NULL	00	SI	15
BELL	07	ESC	27
LF	10	DCS	144
FF	12	CSI	155
CR	13	ST	156
SO	14		

2.8 FOREIGN COMMANDS

Symbols can be defined to create personalized commands that execute as if they were part of the emulator command language. These assignments are called foreign commands.

Example: **NUMSTR:== THIS IS A TEST**
SS :== SHOW SYMBOL
SS NUMSTR

NUMSTR = "THIS IS A TEST"

When the foreign command, SS, is executed from the command line or command file, it is recognized as a foreign command and the symbol value is substituted and executed. The command executed by the command processor is:

SHOW SYMBOL NUMSTR

Up to eight parameters (P1...P8) can be passed to a foreign command. However, in order to process the parameters, the foreign command must execute a command file.

Example: **TYPE :== @DOSTYPE**

Where: The command file DOSTYPE.ECF contains:

```
! ECL FILE TO TYPE A DOS FILE
IF P1 .EQS. "" THEN GOTO ERROR          ! Error if no file
DOS TYPE 'P1'                           ! Type DOS File
EXIT                                    ! Exit
ERROR:
DISPLAY "ERROR - NO FILE SPECIFIED"
EXIT
```

To execute the foreign command to type a DOS file, enter:

TYPE README.TXT

Foreign commands are useful for creating short synonyms for lengthy emulator commands, creating new emulator functions, or changing an emulator command verb to one you like better.

Examples: **KS*END:== KERMIT SEND**
HK:== HELP KEYS
LOGS*CREEN:== LOG/SCREEN/OVERWRITE

Placing an asterisk within a foreign command symbol defines the minimum number of characters that must be entered before it is recognized by the command processor. For example, LOGSCREEN requires that **LOGS** be entered. Additional characters entered thereafter must match the corresponding character in the command exactly.

2.9 LEXICALS

Lexicals are functions that return information about character strings and other items. Lexical functions are not enclosed in quotation marks and often require an argument. Lexicals can be used in expressions in the same manner as character strings, integers, and symbols.

F$EXTRACT

F$EXTRACT(offset,length,string)

Extracts a substring from a string expression.

Arguments:

Offset

An integer value representing a starting position for the extract. Offsets start at 0. The total length of the string, minus one, is the maximum offset value.

Length

An integer value representing the number of characters to extract from the string. A maximum value of 255 can be used to extract the remaining portion of the string.

String

The string expression to extract the substring from.

Return Value: A character string extracted from the argument string.

Example 1:
```
SUBSTR=F$EXTRACT(10,3,"The quick fox jumped.")
SHOW SYMBOL SUBSTR
SUBSTR="fox"
```

Example 2:
```
LAZY = “The quick fox jumped.”
SUBSTR=F$EXTRACT(10,3,LAZY)
SHOW SYMBOL SUBSTR
SUBSTR="fox"
```

F$GETINFO

F$GETINFO(item)

Returns information about the item requested.

Arguments:

Item

The name of the Item to return information about.

Return Value: An integer or string value.

Valid Item Names:

COLOR_SUPPORT Returns TRUE if color support is enabled and FALSE if it is not. Color support is always FALSE if the PC has a monochrome monitor.

("CONNECT") Returns TRUE if the emulator is online (connected). If modem control is disabled when communicating over a COM port, connection status is always true. Connection status is FALSE when the emulator is offline (no connection).

("CONNECT_NAME") Returns the name of the current RS232 or network connection.

Example:
```
DIAL 123-4567
IF F$GETINFO("CONNECT") THEN GOTO LOGIN
...
```
If the modem is connected, the command file jumps to LOGIN label.

F$LENGTH

F$LENGTH(string)

Returns the total number of character in a string.

Arguments:

String
The string expression.

Return Value: An integer value for the length of the string.

Example:
```
TEXT:==This is a test
LEN=F$LENGTH(TEXT)
SHOW SYMBOL LEN
```
LEN = 14 Hex=000E Octal = 000016

F$LOCATE

F$LOCATE(substring,string)

Searches for a character substring within a string and returns the substring's offset. If the substring is not found, the function returns the length of the original string. The first character position is offset 0.

Arguments:

Substring
The character string to search for.

String
The string searched.

Return Value: An integer value representing the offset of the substring argument.

Example 1:
```
TEXT="This was a test"
OFFSET=F$LOCATE(“was”,TEXT)          !Locate “was”
SHOW SYMBOL OFFSET                   !Show the offset when found
```
OFFSET = 5 Hex=0005 Octal = 00005

Example 2:
```
TEXT="This is a test"
OFFSET=F$LOCATE(“TTTT”,TEXT)         !Will not find “TTTT”
SHOW SYMBOL OFFSET                   !Offset=length if not found
```
OFFSET = 14 Hex=000E Octal = 000016

Example 3:
```
! The following example requests a string and prints:
! “THE” FOUND          If “THE” was entered as part of the string.
! “THE” NOT FOUND  If “THE” was not found in the input string.
INQUIRE DATA “ENTER A TEXT STRING: ”
OFFSET=F$LOCATE(“THE”, DATA)
IF F$LENGTH(DATA) .EQ. OFFSET THEN GOTO NOT_FOUND
DISPLAY “”"THE"" FOUND"
EXIT
NOT_FOUND:
DISPLAY “”"THE"" NOT FOUND"
```

F$MESSAGE

F$MESSAGE(status code)

Returns the message string associated with the status code.

Arguments:

Status code

An expression that translates to either a status message mnemonic (e.g.,"INVALARG") or a status message number (e.g.,1248). Using $STATUS or $STATUSID as the status code returns the current error/status message. See Table 8-12 (Error Messages and Status Codes).

Return Value: The complete message string for the status code.

Example:
```
WP XXX
ERROR_MSG=F$MESSAGE($STATUS)
SHOW SYMBOL ERROR_MSG
```
ERROR_MSG="CMD-W-INVKEYW, Invalid qualifier or keyword - XXX"

2.10 DISPLAY LEXICALS

Display lexicals are special lexical functions used with the DISPLAY and INQUIRE commands. Arguments to display lexicals must be strings or string expressions enclosed in parentheses. The display lexicals currently supported are D$BLOCK and D$BOX.

D$BLOCK

D$BLOCK (row, column [,label])

Where: **label** is a symbol or quoted label name.

Displays a block of text. The text block is defined between two block markers **{** and **}**. If the optional label is not provided, the block of text must follow the DISPLAY command (see Form 1). Command execution continues following the end of block marker.

If the optional label is provided, the text block referenced must not lay in the execution path of the command procedure (see Form 2).

Note: Block markers must be on a line by themselves.

Form 1

```
DISPLAY D$BLOCK(10,40)
{
Line one of text.
Line two of text.
}
... next command ...
```

Form 2

```
INQUIRE NAME D$BLOCK(10,40,"LABEL1")
... additional commands ...
EXIT
LABEL1:
{
Line one of text.
}
```

D$BOX

Uses line drawing characters to display a box on the screen.

Form 1 D$BOX (upper left row, column, lower right row, column)

Arguments:

The row and column positions for the upper-left and lower-right corners for the box.

Form 2 D$BOX (row offset, column offset)

Arguments:

The row and column offset for the lower-right corner. The offset is specified relative to the current cursor position. The current cursor position is used for the upper-left corner.

2.11 SYMLEXES

Symlexes are special symbols that function similar to lexicals. They are especially valuable for defining control sequences that require arguments passed to them at run time.

Example: **E$CUP == "<ESC>[$1s;$2sH"**

Defines a Symlex called E$CUP (cursor position control sequence) with 2 string arguments ($1s and $2s) that are passed at run time.

DISPLAY E$CUP(1,1)

Uses E$CUP to position the cursor to row 1, column 1.

2.11.1 Defining a Symlex

Symlexes are defined in the form:

A$A... == "A...$1x...$2x..."

Where: **A** is any alphanumeric character.
A... is one or more alphanumeric characters.
$1 is the first argument.
x is **s** or **n. S** identifies the argument as string. **N** identifies the argument as numeric.
$2 is the second argument (etc.).

Symlex names must have a dollar sign as the second character of the name ($). Any other character can precede or follow the dollar sign. A maximum of eight arguments can be defined in a symlex definition. Each argument must start with a dollar sign and be followed by the argument number and argument type identifier.

When string arguments are substituted at run time, the argument value is passed as a string and quoting is not necessary. If a symlex argument is defined as numeric, it is assumed to be an integer value, symbol, lexical, expression, or quoted string.

If a symlex name is defined that conflicts with a lexical function name, the symlex is ignored. Symlexes can be used wherever symbols or lexicals are accepted.

Example 1: **E$CUP == “<ESC>[$1n;$2sH”**

Defines a global symbol, E$CUP, that sets the cursor to $1n row, $2s column (parameter $1n is defined as numeric and $2s is defined as a string).

DISPLAY E$CUP(“10",30)

Uses the E$CUP symlex to position the cursor to row 10, column 30.

Example 2: **U$DEFKEY == “<ESC>P1;1|$1s/$2n<ESC>\”**

Defines a symlex for loading a VT320 UDK (User-Defined Key). Argument $1s is the key identifier and $2n is the key definition string.

DISPLAY U$DEFKEY(34,"53484f5720555345520d")

Uses the U$DEFKEY symlex to define UDK20 as “SHOW USER<CR>”

Example 3:
```
A$BOLD == “<ESC>[1m”    !Bold Attribute
A$UND == “<ESC>[4m”     !Underline
A$REV == “<ESC>[7m”     !Reverse Video
A$RST == “<ESC>[m”      !Reset Attributes
```

Defines a set of symlexes for setting VT320 video attributes.

DISPLAY A$BOLD + “ BOLD ” + A$RST

Displays the word BOLD in bold and then resets the video attributes.

Example 4:
```
U$DCSWSL == “<CSI>0;3;0|$1n<ST>”
SETUDSL== “DISPLAY U$DCSWSL (”"""USER DEFINED STATUS LINE""")"
```

Defines global symbol SETUDSL to write a string to the user defined status line using the symlex U$DCSWSL. The symlex uses a DCS Private Control Sequence. (Parameter $1n is defined as numeric).

SETUDSL

Writes the string USER DEFINED STATUS LINE to the status line using the symbol U$DCSWSL. The four quotation marks are necessary to send a quoted string to the symlex.

2.12 SYMBOL AND LEXICAL SUBSTITUTION

When processing a command string, the command interpreter performs substitution by replacing the symbol names or lexical functions with their current values.

2.12.1 Automatic Symbol Substitution

In certain contexts, the command interpreter assumes that a string of characters is a symbol name or lexical function. In that case, substitution is automatic and substitution operators are not required or recommended. Automatic symbol substitution takes place under the following contexts:

- On the right side of an = or == assignment statement (but not an := or :== assignment).
- At the beginning of a command line when the symbol is not followed by a symbol assignment operator.
- On arguments for lexical functions.
- On arguments to certain commands such as DISPLAY or WRITE.

Symbols or lexicals in other contexts must be enclosed within a set of substitution operators in order to translate.

2.12.2 Substitution Using Apostrophes

The apostrophe is normally used for symbol substitution. The ampersand is reserved as a special substitution character. See Section 2.12.3 (Ampersands). To substitute a symbol or lexical value, enclose the symbol or lexical name within a set of apostrophes (‘symbol_name’).

If symbol substitution is desired within a quoted string, two apostrophes must be placed in front of the symbol (i.e., “’’symbol’’”) to force substitution.

Example 1:
```
COUNT = 0
TOTAL = COUNT + 1
```
Evaluated as: TOTAL = 0 +1.

Symbol substitution automatically occurs to the right of a symbol assignment statement.

Example 2:
```
SS := SHOW SYMBOL
SS $STATUS
```
Evaluated as: $STATUS = 1 Hex = 0001 Oct = 00001

Symbol substitution occurs automatically on the first word of a command line. SS is defined as a synonym for Show Symbol and is executed as a foreign command.

Example 3:
```
TEXT = “This is it.”
STR = F$EXTRACT(5,2,TEXT)
```
Evaluated as: STR = F$EXTRACT(5,2,"This is it.")

Symbol substitution occurs automatically on any arguments to a lexical function.

Example 4:
```
TOTAL=1
COUNT=2
IF COUNT .EQ. TOTAL THEN GOTO DONE
```
Evaluated as: IF 1 .EQ. 2 THEN GOTO DONE

Symbol substitution in an IF statement. TOTAL and COUNT are both assumed to be symbols. Their values are substituted before evaluating the condition.

Example 5:
```
COUNT = 1
PARAM = P'COUNT'
```
Evaluated as: PARAM = P1

The use of single quotes forces the substitution.

Example 6:
```
FILENAME := X.DAT
STR = "''FILENAME' has been copied."
```
Evaluated as: STR ="X.DAT has been copied."

Symbol substitution is forced by the usage of double, single quotes within the quoted string.

2.12.3 Substitution Using Ampersands

In addition to the apostrophe, the command interpreter recognizes a special substitution operator, the ampersand. The difference between the two is the time when symbol substitution occurs. Symbols preceded by the apostrophe are substituted during phase one; the ampersand is done in phase two. For additional information, refer to the *Three Phases of Symbol Substitution* topic.

In many instances, the apostrophe and ampersand operators are equivalent.

Example:
```
CMD>HELP 'TOPIC'
CMD>HELP &TOPIC
```
These two commands evaluate identically.

However, the following example shows how the results can vary.

Example:
```
CMD>B="XXXXXX"
CMD>A="&B"
CMD>SHOW SYMBOL A
A = "&B"
CMD>DISPLAY 'A'
XXXXXX
CMD>B = "YYYYY"
CMD>DISPLAY 'A'
YYYYY
```
In the first part, SHOW SYMBOL A displays &B because the ampersand is not interpreted within a quoted string. However, &B is interpreted when referenced by the DISPLAY 'A' command. In the second part, B was redefined and the results changed accordingly.

The following restrictions apply to the use of the ampersand:

- It cannot be used within a character string to request symbol substitution.
- It must be preceded by a space or another delimiter.
- It cannot be used to request substitution inside a quoted string.
- To request substitution using the ampersand, append the ampersand to the beginning of the symbol name. Do not use a trailing ampersand.

Ampersands are most effective when used with the apostrophe to affect the order of substitution.

Example: Assume the following symbol definitions: **A:=TRY B:=THIS C:=ONE**
Assume that TEST.ECF contains:
```
COUNT=1
START:SHOW SYMBOL &P'COUNT'
COUNT=COUNT+1
IF COUNT .GT. 3 THEN EXIT
GOTO START
```
This command yields the results:
```
CMD>@TEST A B C
A = "TRY"
B = "THIS"
C = "ONE"
```

The command file displays the values of passed parameters P1 - P3 using the SHOW SYMBOL command. During the phase one of command interpretation, COUNT is replaced by its current value (1 - 3).

By using the ampersand, P'COUNT' (P1 - P3) is substituted in the phase two. Therefore, P1 becomes symbol A, P2 becomes symbol B, and P3 becomes symbol C. The final substitution results in the command lines:

```
SHOW SYMBOL A      (value = TRY)
SHOW SYMBOL B      (value = THIS)
SHOW SYMBOL C      (value = ONE)
```

It is impossible to obtain the above results using the apostrophe substitution character alone. Refer to the following section for more information on the three phases of symbol substitution.

2.12.4 Three Phases of Symbol Substitution

The command interpreter performs symbol substitution in three phases:

Command Input Scanning

In this phase, the interpreter reads the command input and replaces arguments preceded with apostrophes (double apostrophes when strings are enclosed in quotation marks). Symbols preceded by odd groups of apostrophes are translated iteratively. Refer to the *Iterative Substitution Using Apostrophes* topic for more information. Symbols within quoted strings, preceded with double apostrophes, are not translated iteratively.

Command Parsing

During this phase, the command interpreter analyzes the command string and determines whether the first value on the command line is a symbol used as a command synonym (foreign command). If so, the interpreter replaces the symbol with its current value. All substitutions requested with ampersands are performed. In phase two, the Interpreter makes only a single pass through the command string.

Expression Evaluation

During this phase, the command interpreter replaces any remaining symbols used in command expressions. For example, expressions used with the IF command. In phase three, the command interpreter makes only a single pass through the command string.

2.12.4.1 Iterative Substitution Using Apostrophes

When an apostrophe is used to request symbol substitution, the command interpreter performs iterative, or multiple pass, substitution during the first (input scanning) phase of symbol substitution. Iterative substitution is performed from left to right. However, substitution using apostrophes is not iterative when substituting symbols inside quoted strings.

Example:
```
CMD>SYMBOL = "10"
CMD>A ="'SYMBOL'"
CMD>B = 'A'
CMD>SHOW SYMBOL B
B= 10 Hex= 000A Octal= 000012
```

After the statement B = 'A' the resulting integer value of the symbol is 10.

This result is achieved in the following steps:

1) The symbol name A is enclosed in apostrophes, so it is replaced with its current value ('SYMBOL').
2) Because the value ('SYMBOL') is also enclosed in apostrophes, the command interpreter replaces the value SYMBOL with its current value (10).
3) Since value (10) has no apostrophes, the command input scanning phase (phase one) is complete. No further substitution is required during the command parsing or expression evaluation phases. Therefore, 10 is the final value given to the symbol name B. However, note what happens when you define B as:

Example:
```
CMD>B = "'A'"
CMD>SHOW SYMBOL B
B ="'SYMBOL'"
```

In this case, B has the value "'SYMBOL'". The symbol name A is replaced only once, because substitution is not iterative within quoted character strings.

2.12.4.2 Iterative Substitution Using Command Synonyms

The command interpreter performs iterative substitution automatically only when an apostrophe is in the command string. In some cases, you may want to nest synonym definitions.

Example:
```
CMD>COMMAND = "HELP"
CMD>HH = "'COMMAND'"
CMD>HH
CMD-W-INVALCMD, Unrecognized command - 'COMMAND'
```

In this example, when the command synonym HH is processed, the command interpreter performs substitution only once. The resulting string is 'COMMAND'. The command interpreter issues an error message because it cannot detect a command on the line.

The error occurs because, during the first phase of command processing, no substitution is performed (the string HH is not delimited by apostrophes). During the second phase, the string ‘COMMAND’ is substituted for HH because HH is the first value on the command line. No additional substitution is performed.

To correctly use the command synonym HH, it must be enclosed in apostrophes, as shown below:

```
CMD>'HH'
```

In this context, the HH is evaluated during the first phase of command processing because it is delimited by apostrophes. Since the use of apostrophes forces the substitution to be iterative, the resulting value (‘COMMAND’) is also evaluated and the string HELP is substituted in place of ‘HH’.

2.12.4.3 Iterative Substitution in Expressions

When the command interpreter analyzes an expression, any symbols in the expression are replaced only once; iteration is not automatic. However, iterative substitution can be forced by using an apostrophe or an ampersand in the expression. The rules are as follows:

- The command interpreter performs all substitution requested by apostrophes and ampersands before the command string is executed.
- Commands that automatically perform symbol substitution do so after the command string has been processed by the command interpreter.

The following example illustrates iterative substitution in an IF command.

Example: **IF P’COUNT’ .EQS. “” THEN GOTO DONE**

When the command interpreter scans the input line, it replaces the symbol name COUNT with its current value. If the current value of COUNT is 1, the expression is evaluated as follows:

IF P1 .EQS. “” THEN GOTO DONE

Because this string does not have apostrophes, the command interpreter does not perform any additional substitutions. However, when the IF command executes, it automatically evaluates the symbol name P1 and replaces it with its current value.

2.12.4.4 Substitution of Undefined Symbols

If a symbol is not defined when it is used in a command string, the command interpreter either issues an error message or replaces the symbol with a null string, depending on the context. The rules are as follows:

- During command input scanning and during command parsing, the command interpreter replaces all undefined symbols that are preceded by apostrophes or ampersands with null strings.
- During expression evaluation, the command interpreter issues a warning message and does not complete command processing.

2.13 ERROR FACILITY

On completion of a command, a status condition code is saved in the symbol $STATUS to indicate the reason the command terminated. If error handling is enabled, specific error handling actions, based on that reason, are performed. Error handling is enabled by the ON and SET commands. The default conditions are as follows:

- SET ON
- ON ERROR THEN EXIT
- SET ABORT
- ON ABORT THEN STOP
- SET NODEVICE_ERROR
- ON DEVICE_ERROR THEN STOP
- SET NODISCONNECT
- ON DISCONNECT THEN STOP

Note: The default conditions may be modified by a command file.

No action takes place if the error handler for the specific error condition is disabled with one of the following:

- SET NOON
- SET NOABORT
- SET NODEVICE_ERROR
- SET NODISCONNECT

Descriptive error and informational messages issued by the command interpreter break down into four parts:

(1) **facility** (2) **l-** (3) **ident** (4) **text**

The beginning of the message, **facility**, begins with the processor identification letters; EM for the Emulator Processor, CMD for the Command Processor or KER for the Kermit Processor.

The **l** severity level follows:

Table 2-9 Error Message Severity Levels

Level	Definition
E	ERROR
F	FATAL
I	INFO
S	SUCCESS
W	WARNING

Ident is the mnemonic code identifying the message, followed by the **text**.

For example, specifying an invalid command would display an error message:

Example:
```
CMD>DISPLY
CMD-W-INVALCMD, Unrecognized command - DISPLY
```

Once the message displays, the most significant bit (bit 15 of $STATUS) is set to 1, indicating that the message has displayed. The error processor uses this to prevent the message from redisplaying if the status code is passed to the EXIT command. Clearing this bit displays the message again upon exit.

2.13.1 $STATUS Conditional Codes

Error message values are saved as a 16 bit word in the reserved global symbol $STATUS. The breakdown of $STATUS is as follows:

Bits 0-2 Contains the severity level of the message.

Bits 3-14 Contains the message ID number.

Bit 15 Indicates if the error message has displayed.

To correctly identify an error message with bit 15 possibly set, it is necessary to logically AND the $STATUS code with a mask of %X7FFF to ignore bit 15.

The low-order three bits of the $STATUS code are also saved in the reserved global symbol $SEVERITY. These bits represent the severity of the condition that caused the command to terminate. The severity error levels are represented by the following numeric values:

Table 2-10 $STATUS Error Level Severity

Level	Definition
0	WARNING
1	SUCCESS
2	ERROR
3	INFORMATIONAL
4	FATAL

Note: Some severe errors are handled as fatal system errors and cannot be controlled by the user.

The SUCCESS and INFORMATIONAL levels are odd numeric values (true), while the remaining error severity levels are even numeric values (false). This makes it easy to test for successful completion of a command using the IF command.

If the program completes with a SUCCESS numeric value, $STATUS and $SEVERITY is odd and the IF expression is true.

Example 1: **IF .NOT. $STATUS THEN GOTO ERROR**

This IF statement tests the NOT SUCCESS condition of the last executed command.

Example 2: **IF $STATUS THEN DISPLAY “Operation completed successfully”**

This IF statement tests for the SUCCESS condition of the last executed command.

Example 3: **IF ($STATUS .AND. %X7FFF) .EQ. 52 THEN GOTO EXIT_CLEANUP**

This IF statement tests for a specific error message (an Abort).

When the binary status code is stored in $STATUS, the mnemonic value for the error condition is also stored in $STATUSID. The value in $STATUSID can then be tested symbolically for specific errors.

Example 1: **IF $STATUSID .EQS. “EOF” THEN EXIT**

Tests for an EOF condition and then exits if found.

Example 2: **IF $STATUSID .EQS. “SUCCESS” THEN GOTO 100**

Transfers control to label 100 if the previous command was successful.

2.13.2 DOS ERROR LEVEL

To see a listing of the error codes with their DOS ErrorLevel included, execute the following command file:

```
CMD>SET MESSAGE SCREEN
CMD>@ERRMSG
```

2.13.3 Messages

STATUS CODES are made up of three important parts:

Table 2-11 Status Code Description

Title	Description	Found In
L	Severity Level	$SEVERITY
Ident	ID Mnemonic	$STATUSID
Message	Error Message	(see Note)

Note: The message text is not stored in a symbol, however, the message text may be extracted using the F$MESSAGE lexical function:

```
MSG := F$MESSAGE($STATUS)
```

Table 2-12 Error Messages and Status Codes

L	Ident	Message
S	ABORT	>ABORT INTERRUPT<
W	ABORTED	Command process aborted
W	ABSYMD	Abbreviated symbol definition conflict - rename symbol
E	ALREADYCONN	Already connected to node
W	AMBIGCMD	Ambiguous command -
W	AMBIGOPT	Ambiguous option - /
W	AMBKEYW	Ambiguous qualifier or keyword -
W	ARGLENEXC	Argument exceeded maximum length -
S	CMDFONLY	Command or function enabled for command files only
S	CONNLOST	Connection lost
E	DDEBADCONN	DDE Bad conversation handle
E	DDEBADDATA	DDE Bad data handle
E	DDEBADDISC	DDE DISCONNECT failed
E	DDEINVDATAL	Invalid data link requested
E	DDEMAXADVISE	Maximum number of advise items reached
E	DDEMAXCONN	Maximum number of connections reached
E	DDENOCONN	DDE CONNECT failed
E	DDENODATA	DDE Data not available from server
E	DEFNODECONN	Default (auto-connect) node already connected
E	DEFNODEUNDEF	Default (auto-connect) node is undefined
F	DISKFULL	Disk full error
F	DIVBYZERO	Divide by zero error
F	DOSERR	DOS error - unable to execute cmd
E	EOF	End of file detected
W	EXPOVFL	Command line expansion overflow
W	EXPSYN	Invalid expression syntax - check operators and operands
S	FILECREATE	Error creating PC file -
S	FILEOPEN	Error opening PC file -
S	FILEPTR	Error setting file pointer in PC file -
S	FILEREAD	Error reading PC file -
S	FILEUPDATE	Error updating PC file -
S	FILEWRITE	Error writing PC file -
E	GRAPHICSNOTLOADED	Graphics not loaded
E	HELPREAD	Error reading HELP file - data not properly formatted
E	INSFMEM	Insufficient DOS memory
W	INVALARG	Invalid argument -
W	INVALBAUD	Invalid Baud Rate for INT 14 Redirection
W	INVALCMD	Unrecognized command -
W	INVALDECTOKSTR	Invalid DEC TOKEN string

Table 2-12 Error Messages and Status Codes (cont'd)

L	Ident	Message
W	INVALOPT	Invalid option - /
W	INVALTOK	Invalid TOKEN code -
S	INVFSPEC	Invalid PC file specification -
W	INVKEYW	Invalid keyword or qualifier -
W	INVOPER	Unrecognized operator in expression -
E	INVSKEY	Invalid Softkey
W	IVDELTIM	Invalid delta time argument -
W	IVFNAM	Invalid LEXICAL or SYMLEX name -
S	IVSETUP	Invalid SETUP file name -
W	IVSYMLVAR	Invalid SYMLEX variable
S	KHOSTERR	Error packet received from HOST
S	KPROTO	Protocol error
I	KRENAME	File exists - could not rename
S	KRETRY	Packet retry count exceeded
S	KTIMOUT	Timed out waiting for packet
E	LASTNODECONN	Last node already connected
E	LASTNODEUNDEF	Last node is undefined
E	LINELONG	Command line exceeds maximum length
W	LOGFEXIST	Log file already exists - use /OVERWRITE or /APPEND option
W	LOGICDEF	Logical name already defined -
W	LOGINPROG	Logging in progress - request ignored
W	MISKEYW	Missing keyword or qualifier
W	MISOPTPAR	Missing option parameter - check options for required arguments
S	NETABORTED	Connection aborted
S	NETADDNAM	Error adding name to network
I	NETCONNBAPI	BAPI node connected
I	NETCONNCOM	COM port connected
I	NETCONNCTERM	CTERM node connected
I	NETCONNECT	Connecting to Network
E	NETCONNERR	Error attempting connection
E	NETDISCON	Session disconnected
I	NETINVCOM	COM port number invalid
E	NETINVPASS	Invalid password
I	NETINVPORT	Port number invalid

Table 2-12 Error Messages and Status Codes (cont'd)

L	Ident	Message
I	NETNOCOM	COM port not specified
E	NETNONFS	NFS is not installed
I	NETNOSESS	Multi-sessions not enabled
I	NETNOTCONN	Session not connected
E	NETNOWSK	WINSOCK network is not installed
W	NETONLY	Only available on network versions
I	NETSESSMAX	No more sessions available
E	NETUNKNOWN	Requested node is unknown
E	NETUNREACH	Node is currently unreachable
E	NODENAMEREQD	Node name is required for connection
E	NOLABEL	GOTO label not found -
W	NOMSG	Message number not found - %X
W	NOMSGID	Message identifier not found -
E	NORETURN	No RETURN pointer found from prior GOSUB command
E	NOTCONN	Not connected to a port
E	NOTEXTBLK	DISPLAY text block not found
W	NOTHEN	IF or ON statement syntax error - check placement of THEN keyword
E	PICFILEEXISTS	Picture file already exists
E	PICFILENOCREATE	Cannot create picture file
E	PICFILENOEXIST	Picture file does not exist
F	PROGERR	Program check error - contact technical support for assistance
S	PRTNOTRDY	Printer not ready
W	READTIMOUT	Read time out error
S	SETFOPEN	Error opening Setup File -
S	SETFREAD	Error reading Setup File -
S	SETFVER	Setup File Version Error -
S	SETFWRITE	Error writing Setup File -
I	SYMTRUNC	Symbol truncated to -
W	SYNTAX	Command syntax error
w	UNDEFSYM	Undefined symbol -
w	UNDFILE	PC file not open, check logical filename -
I	UNDLOGIC	Undefined logical -
W	VALOVFL	Value overflow
W	WILDCARD	Improper use of wildcards for this command or expression
F	WINERR	WINDOWS error
E	XFERERROR	Unidentified File Transfer Error

CHAPTER 9
VT320 PROGRAMMING

OVERVIEW

This chapter describes the character encoding concepts for the VT320. It covers control functions (control characters, escape sequences, and device control strings). Control functions are used in a program to specify how the emulator processes, sends and displays characters. Each control function has a unique name and each name has a unique, mnemonic abbreviation.

3.1 QUICK REFERENCE TABLES

This section contains quick reference tables for each of the main areas of programming information, namely: character sets, transmitted codes, received codes and reports. A separate section for each area contains more detailed information.

3.1.1 Character Sets

Table 3-1 Character Set Quick Reference

Designating Character Sets

$^{E}S_{C}$ Intermediate Final

Intermediate				Final	
94 Character Sets		**96 Character Sets**			
To Select	Use	To Select	Use	To Select	Use
G0	(	G1	-	ASCII	B
G1	)	G2	.	DEC Supplemental Graphic	%5
G2	*	G3	/	ISO Latin-1	A
G3	+			User-preferred supplemental	<
				DEC Special Graphic	0

Mapping Character Sets

Locking Shifts	
Code	**Function**
^{S}I	Locking shift 0. Maps G0 into GL
^{S}O	Locking shift 1. Maps G1 into GL
$^{E}S_{C}$ ~	Locking shift 1, right. Maps G1 into GR *
$^{E}S_{C}$ n	Locking shift 2. Maps G2 into GL *
$^{E}S_{C}$ }	Locking shift 2, right. Maps G2 into GR *
$^{E}S_{C}$ o	Locking shift 3. Maps G3 into GL *
$^{E}S_{C}$ |	Locking shift 3, right. Maps G3 into GR *

* Indicates VT300 mode only

Single Shifts		
8-Bit Code	**7-Bit Code**	**Function**
$^{S}S_{2}$	$^{E}S_{C}$ N	Single Shift 2. Maps G2 into GL for the next character.
$^{S}S_{3}$	$^{E}S_{C}$ O	Single Shift 3. Maps G3 into GL for the next character.

3.1.2 Transmitted Codes

Table 3-2 Transmitted Codes Quick Reference

Key	Code	
Editing Keypad		
Find	$^{C}S_{I}$ 1~	
Insert Here	$^{C}S_{I}$ 2~	
Remove	$^{C}S_{I}$ 3~	
Select	$^{C}S_{I}$ 4~	
Prev Screen	$^{C}S_{I}$ 5~	
Next Screen	$^{C}S_{I}$ 6~	
Cursor Keys	**Reset Normal**	**Set Application**
	$^{C}S_{I}$ A	$^{S}S_{3}$ A
	$^{C}S_{I}$ B	$^{S}S_{3}$ B
	$^{C}S_{I}$ C	$^{S}S_{3}$ C
	$^{C}S_{I}$ D	$^{S}S_{3}$ D
Auxiliary Keypad	Numeric	**Application**
0	0	$^{S}S_{3}$ p
1	1	$^{S}S_{3}$ q
2	2	$^{S}S_{3}$ r
3	3	$^{S}S_{3}$ s
4	4	$^{S}S_{3}$ t
5	5	$^{S}S_{3}$ u
6	6	$^{S}S_{3}$ v
7	7	$^{S}S_{3}$ w
8	8	$^{S}S_{3}$ x
9	9	$^{S}S_{3}$ y
,	(minus)	$^{S}S_{3}$ m
.	(comma)	$^{S}S_{3}$ l
PF1	(period)	$^{S}S_{3}$ n
PF2	$^{S}S_{3}$ P	$^{S}S_{3}$ P
PF3	$^{S}S_{3}$ Q	$^{S}S_{3}$ Q
PF4	$^{S}S_{3}$ R	$^{S}S_{3}$ R
Enter	$^{S}S_{3}$ S	$^{S}S_{3}$ S
	^{C}R or $^{C}R^{L}F$	$^{S}S_{3}$ M

Table 3-2 Transmitted Codes Quick Reference (cont'd)

Key	Code
Top Row Function Keys	
Hold Screen (F1)	*
Print Screen (F2)	*
Set-Up (F3)	*
Data/Talk (F4)	*
Break (F5)	*
F6	$^{C}S_{I}$ 17 ~
F7	$^{C}S_{I}$ 18 ~
F8	$^{C}S_{I}$ 19 ~
F9	$^{C}S_{I}$ 20 ~
F10	$^{C}S_{I}$ 21 ~
F11	$^{C}S_{I}$ 23 ~
F12	$^{C}S_{I}$ 24 ~
F13	$^{C}S_{I}$ 25 ~
F14	$^{C}S_{I}$ 26 ~
Help (F15)	$^{C}S_{I}$ 28 ~
Do (F16)	$^{C}S_{I}$ 29 ~
F17	$^{C}S_{I}$ 31 ~
F18	$^{C}S_{I}$ 32 ~
F19	$^{C}S_{I}$ 33 ~
F20	$^{C}S_{I}$ 34 ~

* Indicates that codes are not generated.

3.1.3 Received Codes

3.1.3.1 VT320 Control Sequences

Table 3-3 VT320 Control Sequences

Escape Sequence	Function
Set Character Attributes	
$^{C}S_{I}$ Ps;... m	Character attributes
Ps = 0	all attributes off
Ps = 1	bold on
Ps = 4	underscore on
Ps = 5	blink on
Ps = 7	reverse video on
Ps = 2 2	normal intensity
Ps = 2 4	not underscored
Ps = 2 5	not blinking
Ps = 2 7	positive image
$^{C}S_{I}$ " q	All Non-graphic off
$^{C}S_{I}$ 0 " q	All Non-graphic off
$^{C}S_{I}$ 1 " q	All Non-erasable on
$^{C}S_{I}$ 2 " q	All Non-erasable off
Compatibility Level	
$^{C}S_{I}$ 61"p	Level 1 (VT100)
$^{C}S_{I}$ 62"p	Level 3 (VT300 8-bit)
$^{C}S_{I}$ 62;0"p	Level 3 (VT300 8-bit)
$^{C}S_{I}$ 62;1"p	Level 3 (VT300 7-bit)
$^{C}S_{I}$ 62;2"p	Level 3 (VT300 8-bit)
$^{C}S_{I}$ 63"p	Level 3 (VT300 8-bit)
$^{C}S_{I}$ 63;0"p	Level 3 (VT300 8-bit)
$^{C}S_{I}$ 63;1"p	Level 3 (VT300 7-bit)
$^{C}S_{I}$ 63;2"p	Level 3 (VT300 8-bit)
Cursor Positioning	
$^{C}S_{I}$ Pn A	Cursor up
$^{C}S_{I}$ Pn B	Cursor down
$^{C}S_{I}$ Pn C	Cursor right
$^{C}S_{I}$ Pn D	Cursor left
$^{C}S_{I}$ Pl;Pc H	Direct cursor addressing
$^{C}S_{I}$ Pl;Pc f	Direct cursor addressing
$^{C}S_{I}$ H	Home

Table 3-3 VT320 Control Sequences (cont'd)

Escape Sequence	Function
Cursor Movement (cont'd)	
$^{C}_{S_I}$ f	Home
$^{I}_{N_D}$	Index
$^{E}_{S_C}$ D	Index
$^{N}_{E_L}$	New Line
$^{E}_{S_C}$ E	New Line
$^{R}_{I}$	Reverse Index
$^{E}_{S_C}$ M	Reverse Index
Editing	
$^{C}_{S_I}$ Pn P	Delete Pn characters
$^{C}_{S_I}$ Pn @	Insert Pn characters
$^{C}_{S_I}$ Pn L	Insert Pn lines
$^{C}_{S_I}$ Pn M	Delete Pn lines
Erasing	
$^{C}_{S_I}$ Pn X	Erase next Pn characters from cursor
$^{C}_{S_I}$ K	Cursor to end of line
$^{C}_{S_I}$ 0 K	Cursor to end of line
$^{C}_{S_I}$ 1 K	Beginning of line to cursor
$^{C}_{S_I}$ 2 K	Entire line
$^{C}_{S_I}$ J	Cursor to end of screen
$^{C}_{S_I}$ 0 J	Cursor to end of screen
$^{C}_{S_I}$ 1 J	Beginning screen to cursor
$^{C}_{S_I}$ 2 J	Erase entire screen
$^{C}_{S_I}$? K	Selective erase from cursor to end of line
$^{C}_{S_I}$? 0 K	Selective erase from cursor to end of line
$^{C}_{S_I}$? 1 K	Selective erase from beginning of line to cursor
$^{C}_{S_I}$? 2 K	Selective erase entire line
$^{C}_{S_I}$? J	Selective erase from cursor to end of screen
$^{C}_{S_I}$? 0 J	Selective erase from cursor to end of screen
$^{C}_{S_I}$? 1 J	Selective erase from top of screen
$^{C}_{S_I}$? 2 J	Selective erase entire screen
Line Attributes	
$^{E}_{S_C}$ #3	Double height - top half
$^{E}_{S_C}$ #4	Double height - bottom half
$^{E}_{S_C}$ #5	Single width - single height
$^{E}_{S_C}$ #6	Double width - single height

Table 3-3 VT320 Control Sequences (cont'd)

Escape Sequence		Function
Terminal Modes		
Set	**Reset**	**Mode Name**
C_{S_I} 2h	C_{S_I} 2l	Keyboard Action mode
C_{S_I} 4h	C_{S_I} 4l	Insert/Replace mode
C_{S_I} 12h	C_{S_I} 12l	Send/Receive mode
C_{S_I} 20h	C_{S_I} 20l	Line feed/new line
C_{S_I} ?1h	C_{S_I} ?1l	Cursor key mode
	C_{S_I} ?2l	VT52 mode
Set	**Reset**	**Mode Name**
C_{S_I} ?3h	C_{S_I} ?3l	Column mode
C_{S_I} ?4h	C_{S_I} ?4l	Scrolling mode
C_{S_I} ?5h	C_{S_I} ?5l	Screen mode
C_{S_I} ?6h	C_{S_I} ?6l	Origin mode
C_{S_I} ?7h	C_{S_I} ?7l	Auto Wrap mode
C_{S_I} ?8h	C_{S_I} ?8l	Auto repeat
C_{S_I} ?18h	C_{S_I} ?18l	Form Feed mode
C_{S_I} ?19h	C_{S_I} ?19l	Screen Print mode
C_{S_I} ?25h	C_{S_I} ?25l	Text cursor mode
C_{S_I} ?42h	C_{S_I} ?42l	Character Set mode
C_{S_I} ?66h	C_{S_I} ?66l	Numeric keypad
C_{S_I} ?67h	C_{S_I} ?67l	Backarrow key
C_{S_I} Ps $}$		Select status display
Ps = 0		main display
Ps = 1		status line
C_{S_I} Ps $~$		Select status line type
Ps = 0		none
Ps = 1		indicator
Ps = 2		host-writable
E_{S_C} =	E_{S_C} >	Keypad mode
Printing		
C_{S_I} i		Print Screen
C_{S_I} 0i		Print Screen
C_{S_I} 4i		Print Controller mode off
C_{S_I} 5i		Print Controller mode on
C_{S_I} ?1i		Print Cursor Line
C_{S_I} ?4i		Auto Print mode off
C_{S_I} ?5i		Auto Print mode on

Table3-3 VT320 Control Sequences (cont'd)

Escape Sequence	Function
Programmable LEDs	
$^{C}S_{I}$ Ps;Ps q	Programmable LEDs
Ps = 0	all LEDs off
Ps = 1	L1 on
Ps = 2	L2 on
Ps = 3	L3 on
Ps = 4	L4 on
Terminal Reset Mode	
$^{E}S_{C}$ c	Hard terminal reset
$^{C}S_{I}$!p	Soft terminal reset
Scrolling Region	
$^{C}S_{I}$ Pt; Pb r	Define scroll region
Select C1 Control Transmission	
$^{E}S_{C}$ space F	7-bit C1 control transmission
$^{E}S_{C}$ space G	8-bit C1 control transmission
Tab Stops	
$^{H}T_{S}$	Set tab at current column
$^{E}S_{C}$ H	Set tab at current column
$^{C}S_{I}$ g	Clear at current column
$^{C}S_{I}$ 0 g	Clear at current column
$^{C}S_{I}$ 3 g	Clear all tabs
User Defined Keys (DECUDK)	
$^{D}C_{S}$ Pc;Pl | ky1/st1;ky2/st2;...kyn/stn ^{S}T	
DCS Private Sequences	
$^{C}S_{I}$ 0;0|	Enable Status Line
$^{C}S_{I}$ 0;1|	Disable Status Line
$^{C}S_{I}$ 0;2|	Erase Status Line
$^{C}S_{I}$ 0;3;Pc|...String... ^{S}T	Write Status Line
$^{C}S_{I}$ 2;n|	Set/Reset Local Echo
$^{C}S_{I}$ 3;n|	Set/Reset WP mode
$^{C}S_{I}$ 4;pl|	Set Printer Port
$^{C}S_{I}$ 5|...Command String... ^{S}T	Do Emulator Command
$^{C}S_{I}$ 6|	Request Product ID
$^{C}S_{I}$ 7;pl|	Set number of lines per screen

3.1.3.2 VT100 Escape Sequences

Table 3-4 VT100 Escape Sequences

Escape Sequence		Function
Character Attributes		
ESC [Ps;... m		Character attributes
Ps = 0		All attributes off
Ps = 1		Bold on
Ps = 4		Underscore on
Ps = 5		Blink on
Ps = 7		Reverse video on
Character Sets		
G0	**G1**	
ESC (A	ESC) A	UK set
ESC (B	ESC) B	US ASCII set
ESC (0	ESC) 0	Special Graphics set
ESC (1	ESC) 1	Alternate ROM
ESC (2	ESC) 2	Alternate ROM Special Graphics set
Cursor Movement		
ESC [Pn A		Cursor up
ESC [Pn B		Cursor down
ESC [Pn C		Cursor right
ESC [Pn D		Cursor left
ESC [Pl;Pc H		Direct cursor addressing
ESC [Pl;Pc f		Direct cursor addressing
ESC D		Index
ESC E		New line
ESC M		Reverse index
ESC 7		Save cursor - attributes
ESC 8		Restore cursor - attributes
Erase		
ESC [K		Cursor to end of line
ESC [0 K		Cursor to end of line
ESC [1 K		Beginning of line to cursor
ESC [2 K		Entire line
ESC [J		Cursor to end of screen
ESC [0 J		Cursor to end of screen
ESC [1 J		Beginning screen to cursor
ESC [2 J		Erase entire screen
Line Size		
ESC #3		Double height - top half
ESC #4		Double height - bottom half

Table 3-4 VT100 Escape Sequences (cont'd)

Escape Sequence		Function
Line Size (cont'd)		
$^{E}_{SC}$ #5		Single width - single height
$^{E}_{SC}$ #6		Double width - single height
Modes		
Set	**Reset**	Mode Name
$^{E}_{SC}$[20h	$^{E}_{SC}$[20l	Line feed/new line
$^{E}_{SC}$[?1h	$^{E}_{SC}$[?1l	Cursor key mode
$^{E}_{SC}$[?3h	$^{E}_{SC}$[?3l	Column mode
$^{E}_{SC}$[?4h	$^{E}_{SC}$[?4l	Scrolling mode
$^{E}_{SC}$[?5h	$^{E}_{SC}$[?5l	Screen mode
$^{E}_{SC}$[?6h	$^{E}_{SC}$[?6l	Origin mode
$^{E}_{SC}$[?7h	$^{E}_{SC}$[?7l	Wraparound
$^{E}_{SC}$[?8h	$^{E}_{SC}$[?8l	Auto repeat
$^{E}_{SC}$[?9h	$^{E}_{SC}$[?9l	Interlace
$^{E}_{SC}$ 1	$^{E}_{SC}$ 2	Graphic process option
$^{E}_{SC}$ =	$^{E}_{SC}$ >	Keypad mode
Programmable LEDs		
$^{E}_{SC}$ [Ps;Ps q		Programmable LEDs
Ps = 0		All LEDs off
Ps = 1		L1 on
Ps = 2		L2 on
Ps = 3		L3 on
Ps = 4		L4 on
Reset		
$^{E}_{SC}$ c		Reset
Scrolling Region		
$^{E}_{SC}$ [Pt; Pb r		Define scroll region
Tab Stops		
$^{E}_{SC}$ H		Set tab at current column
$^{E}_{SC}$ [g		Clear at current column
$^{E}_{SC}$ [0 g		Clear at current column
$^{E}_{SC}$ [3 g		Clear all tabs

3.1.3.3 VT52 Escape Sequences

Table 3-5 VT52 Escape Sequences

Escape Sequence	Function
^{ESC} A	Cursor up
^{ESC} B	Cursor down
^{ESC} C	Cursor right
^{ESC} D	Cursor left
^{ESC} F	Enter graphics mode
^{ESC} G	Exit graphics mode
^{ESC} H	Cursor to home position
^{ESC} I	Reverse line feed
^{ESC} J	Erase to end of screen
^{ESC} K	Erase to end of line
^{ESC} Y	Direct cursor address
^{ESC} Z	Identify
^{ESC} =	Enter alternate keypad mode
^{ESC} >	Exit alternate keypad mode
^{ESC} <	Enter ANSI mode
^{ESC} ^	Enter auto print mode
^{ESC} _	Exit auto print mode
^{ESC} W	Enter printer controller mode
^{ESC} X	Exit printer controller mode
^{ESC}]	Print screen
^{ESC} V	Print cursor line

3.1.4 Reports

3.1.4.1 VT320 Reports

Table 3-6 VT320 Reports

Host Directives* (host to Emulator)	Reports (Emulator to host)	Function
CSI c or **CSI 0 c**	**CSI ? Psc ; Ps1 ; ...Psn c** **Psc** Operating level 1 level 1 (VT100) 6 level 1 (VT102) 62 level 2 (VT200) 63 level 3 (VT300) **Ps1...Psn** Extensions 1 132 columns 2 printer port 6 selective erase 7 soft character set 8 user-defined keys 9 NRC set	**Primary Device Attributes**
CSI > c or **CSI > 0 c**	**CSI > Pp ; Pv ; Po c** **Pp** Identification code 24 VT320 **Pv** Firmware version **Po** Hardware options 0 no options	**Secondary Device Attributes**
CSI 6 n	**CSI Pl ; Pc R** **Pl** Line number **Pc** Column number	**Device Status Reports** Cursor Position
CSI ? 26 n	**CSI ? 27 ; Pd n** **Pd** Keyboard dialect 1 North American	Keyboard Dialect
CSI 5 n	**CSI 0 n** no malfunction **CSI 3 n** malfunction	Operating Status
CSI ? 15 n	**CSI ? 13 n** no printer **CSI ? 10 n** printer ready **CSI ? 11 n** printer not ready	Printer Status

Table 3 -6 VT320 Reports (cont'd)

Host Directives* (host to Emulator)	Emulator Reports (Emulator to host)	Function
$CSI ? 25 n	**$CSI ? 20 n** UDKs unlocked **$CSI ? 21 n** UDKs locked	UDK Status (VT300 mode only)
$CSI Ps $ u **Ps** Report requested 0 ignored 1 terminal state report	**$DCS 1 $ s D...D** **<checksums 1 and 2> $ST** **D...D** Report data	**Terminal State Reports** (VT300 mode only)
$DCS Ps $ p D...D $ST **Ps** Data string format 0 error 1 terminal state report **D...D** Restored data		Restore terminal state
		Presentation State Reports (VT300 mode only)
$CSI Ps $ w **Ps** Report requested 0 error 1 cursor information report 2 tab stop report	**$DCS 1 $ u D...D $ST** **D...D** Data string **$DCS 2 $ u D...D $ST** **D...D** Tab stops	Cursor information report Tab stop report
$DCS Ps $ t D...D $ST **Ps** Data string format 0 error 1 cursor information report 2 tab stop report **D...D** Data string		Restore presentation state
$CSI Pa $ p **Pa** ANSI mode	**$CSI Pa ; Ps $ y** **Pa** ANSI mode **Ps** Mode state 0 unknown state 1 set 2 reset 3 permanently set 4 permanently reset	**Mode Settings** (VT300 mode only)

Table 3 -6 VT320 Reports (cont'd)

Host Directives* (host to Emulator)	Emulator Reports (Emulator to host)	Function
CS_I ? Pd $ p **Pd** DEC private mode	**CS_I ? Pd ; Ps $ y** **Pd** DEC private mode **Ps** Mode state 0 unknown state 1 set 2 reset 3 permanently set 4 permanently reset	
CS_I Pa ; ...Pa h **Pa** ANSI mode		Set mode
CS_I ?Pd ; ...Pd h **Pd** DEC private mode		
CS_I Pa ; ...Pa l **Pa** ANSI mode		Reset mode
CS_I ? Pd ;... Pd l **Pd** DEC private mode		
		Cursor Settings
ES_C 7		Save cursor
ES_C 8		Restore cursor
DCS $ q D...D ST **D...D** Intermediate and/or final characters of function.	**DCS Ps $ r D...D ST** **Ps** Request validity 0 invalid request 1 valid request **D...D** Intermediate and/or final characters of function.	**Control Function Settings** (VT300 mode only)
CS_I & u	**DCS 0 ! u % 5 ST** DEC Supplemental Graphic **DCS 1 ! u A ST** ISO Latin-1 Supplemental	**User-preferred Supplemental Set** (VT300 mode only)

Table 3-7 ANSI Modes

Pa	Mode
2	Keyboard action
3	Control representation *
4	Insert/replace
10	Horizontal editing
12	Send/receive
20	Line feed/new line

* Control representation is not supported.

Table 3-8 DEC Private Modes

Pd	Mode	Pd	Mode
1	Cursor keys	18	Print form feed
2	ANSI	19	Printer extent
3	Column	25	Text cursor enable
4	Scrolling	42	National Replacement Character set
5	Screen	66	Numeric keypad
6	Origin	67	Backarrow key
7	Autowrap	68	Keyboard usage *
8	Autorepeat		

* Keyboard usage is not supported and is permanently reset.

3.1.4.2 VT100 Reports

Table 3-9 VT100 Reports

Host Directives (host to Emulator)	Emulator Reports (Emulator to host)	Function
E_{SC} [6 n	**E_{SC} [Pl;Pc R**	Cursor Position
E_{SC} [c or **E_{SC} [0 c**	**E_{SC} [?1; Ps c**	Status Report
E_{SC} Z	**E_{SC} [?1; Ps c**	Terminal Identification
	Ps Identification Code	
	0 Base VT100	
	1 STP	
	2 AVO	
	3 AVO and STP	
	4 GPO	
	5 GPO and STP	
	6 GPO and AVO	
	7 GPO, STP, and AVO	

3.2 CHARACTER ENCODING

The VT320 uses an 8-bit character encoding scheme and a 7-bit code extension technique that are compatible with ANSI (American National Standards Institute) standards.

When operating in VT100 or VT52 mode, you are limited to working in a 7-bit environment. There are three requirements for operating in an 8-bit environment:

- Communications must be set for 8-bits and no parity.
- Your program must be 8-bit compatible.
- The emulator must be in VT320, 7-bit or 8-bit mode.

VT320 7-bit mode displays the VT320 8-bit character set while sending 7-bit control sequences to the host.

VT320 8-bit mode also displays the 8-bit character set, but sends 8-bit control sequences to the host.

Note: VT320 8-bit mode is not a communication setting. It is an operating environment. To select 8-bit communications, configure the emulator to No Parity.

3.2.1 7-Bit ASCII Codes

The 7 Bit ASCII Code table shows the octal, decimal, and hexadecimal code for each 7-bit ASCII character.

Table 3-10 7-Bit ASCII Codes

3.2.2 8-Bit ASCII Codes

The 8-Bit ASCII Codes table able 3 -11 shows the 8-bit code table, which has twice as many code values as the 7-bit code table.

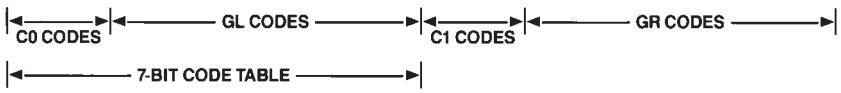
All codes on the left half of the 8-bit table (columns 0 through 7) are 7-bit compatible; the 8th bit is not set, and can be ignored or assumed to be 0. You can use these codes in a 7-bit or an 8-bit environment. All codes on the right half of the table (columns 8 through 15) have their 8th bit set. You can only use these codes in an 8-bit environment.

The 8-bit code table has two sets of control characters, C0 (control 0) and C1 (control 1). The table also has two sets of graphics characters, GL (graphic left) and GR (graphic right).

The basic functions of the C0 and C1 codes are defined by ANSI. The C0 codes are 7-bit compatible. The C1 codes represent 8-bit control characters that perform functions beyond those possible with the C0 codes. You can only use C1 codes in an 8-bit environment.

Table 3-11 8-Bit ASCII Codes

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00	NUL	DLE	SP							DCS	///					
01	SOH	DC1								PU1						
02	STX	DC2								PU2						
03	ETX	DC3								STS						
04	EOT	DC4							IND	CCH						
05	ENQ	NAK							NEL	MW						
06	ACK	SYN							SSA	SPA						
07	BEL	ETB							ESA	EPA						
08	BS	CAN							HTS							
09	HT	EM							HTJ							
10	LF	SUB							VTS							
11	VT	ESC							PLD	CSI						
12	FF	FS							PLU	ST						
13	CR	GS							RI	OSC						
14	SO	RS							SS2	PM						
15	SI	US						DEL	SS3	APC						///



3.2.3 Control Functions

Control functions are a set of instructions used to program the terminal emulator. All control functions can be expressed in single-byte or multi-byte codes.

Single-byte codes are the C0 and C1 control characters. You can perform a limited number of functions using C0 characters. A few more functions are available using C1 characters, but they must be used in an 8-bit environment.

Multi-byte control codes represent far more functions than single-byte codes, due to the variety of code combinations possible. These codes are called control sequences, escape sequences, and device control strings.

3.2.3.1 Control Sequences

A control sequence starts with a $^{C}S_{I}$ (Control Sequence Introducer), followed by one or more ASCII characters. The 8-bit $^{C}S_{I}$ can also be expressed as the 7-bit equivalent $^{E}S_{C}$ [(for use in a 7-bit environment). Thus, you can express all control sequences as escape sequences where the second character is the [. For example, the following two sequences are equivalent and perform the same function (they change the display from 80 columns to 132 columns).

$^{C}S_{I}$? 3 h
$^{E}S_{C}$ [? 3 h

Since the 8-bit $^{C}S_{I}$ uses one less byte than the 7-bit equivalent, $^{E}S_{C}$ [, you will gain processing speed by using the $^{C}S_{I}$. However, you can only use a sequence starting with the $^{C}S_{I}$ character in an 8-bit environment.

You can express any C1 control character as a two character escape sequence whose second character has a code that is 40 (hexadecimal) less than that of the C1 character. For example, ^{S}T is the same as $^{E}S_{C}$ \.

3.2.3.2 Escape Sequences

All escape sequences start with the same C0 character, $^{E}S_{C}$, and are followed by one or more ASCII characters. For example, the following escape sequence causes the current line to have double-width characters:

$^{E}S_{C}$ # 6

Because escape sequences use only 7-bit characters, you can use them in 7-bit or 8-bit environments.

You can make any escape sequence whose second character is in the range of column-4, row-0 through column-5, row-15 (refer to the 7-Bit ASCII Codes topic for more information) one byte shorter by removing the $^{E}S_{C}$ and adding 40 (hexadecimal) to the code of the second character. This generates a C1 control character.

3.2.3.3 Device Control Strings

A device control string ($^{D}C_{S}$) is a delimited string of characters used in a data stream as a logical entity for control purposes. It consists of an opening delimiter (a device control string introducer), a command string (data) and a closing delimiter (a string terminator).

Device control strings are used to download character sets and to load user-defined keys.

Table 3-12 Device Control String

Device Control String	Data	String Terminator
$^{D}C_{S}$	UDKs or Character Set	$^{S}_{T}$

A **device control character** ($^{D}C_{S}$) is an 8-bit control character. It is expressed as $^{E}S_{C}$ P when coding for a 7-bit environment.

A **string terminator** ($^{S}_{T}$) is also an 8-bit control character. It is expressed as $^{E}S_{C}$ \ when coding for a 7-bit environment.

3.3 CHARACTER SETS

Although the C0 and C1 function codes cannot be changed, the GL and GR codes can have different character sets mapped into them. The Mapping Character Sets topic describes the commands for mapping character sets into GL or GR.

The emulator supports the following character sets:

- DEC Multinational (consists of ASCII and DEC Supplemental Character sets)
- ISO Latin-1
- DEC Special Graphics
- National Replacement Character
- Downloadable

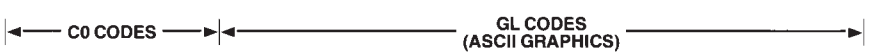
3.3.1 DEC Multinational

The DEC multinational character set is the default character set.

The C0 and GL codes are the ASCII control codes and character set. The C1 and GR codes are the DEC multinational 8-bit control characters and character set. The C1 and GR control codes and characters are not available in VT52 and VT100 modes.

Table 3-13 DEC Multinational Character Set

	0		1		2		3		4		5		6		7	
0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	`	140 96 60	p	160 112 70
1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	a	141 97 61	q	161 113 71
2	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	b	142 98 62	r	162 114 72
3	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	c	143 99 63	s	163 115 73
4	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	d	144 100 64	t	164 116 74
5	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	e	145 101 65	u	165 117 75
6	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	f	146 102 66	v	166 118 76
7	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	g	147 103 67	w	167 119 77
8	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	h	150 104 68	x	170 120 78
9	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	i	151 105 69	y	171 121 79
10	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	j	152 106 6A	z	172 122 7A
11	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	k	153 107 6B	{	173 123 7B
12	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	l	154 108 6C	|	174 124 7C
13	CR	15 13 D	GS	35 29 1D	–	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	m	155 109 6D	}	175 125 7D
14	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	n	156 110 6E	~	176 126 7E
15	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	_	137 95 5F	o	157 111 6F	DEL	177 127 7F



33	OCTAL
27	DECIMAL
1B	HEX

Table 3-13 DEC Multinational Character Set (cont'd)

8		9		10		11		12		13		14		15		
	200 128 80	DCS	220 144 90		240 160 A0	°	260 176 B0	À	300 192 C0		320 208 D0	à	340 224 E0		360 240 F0	0
	201 129 81	PU1	221 145 91	¡	241 161 A1	±	261 177 B1	Á	301 193 C1	Ñ	321 209 D1	á	341 225 E1	ñ	361 241 F1	1
	202 130 82	PU2	222 146 92	¢	242 162 A2	2	262 178 B2	Â	302 194 C2	Ò	322 210 D2	â	342 226 E2	ò	362 242 F2	2
	203 131 83	STS	223 147 93	£	243 163 A3	3	263 179 B3	Ã	303 195 C3	Ó	323 211 D3	ã	343 227 E3	ó	363 243 F3	3
IND	204 132 84	CCH	224 148 94		244 164 A4		264 180 B4	Ä	304 196 C4	Ô	324 212 D4	ä	344 228 E4	ô	364 244 F4	4
NEL	205 133 85	MW	225 149 95	¥	245 165 A5	μ	265 181 B5	Å	305 197 C5	Õ	325 213 D5	å	345 229 E5	õ	365 245 F5	5
SSA	206 134 86	SPA	226 150 96		246 166 A6	¶	266 182 B6	Æ	306 198 C6	Ö	326 214 D6	æ	346 230 E6	ö	366 246 F6	6
ESA	207 135 87	EPA	227 151 97	§	247 167 A7	·	267 183 B7	Ç	307 199 C7	Œ	327 215 D7	ç	347 231 E7	œ	367 247 F7	7
HTS	210 136 88		230 152 98	¤	250 168 A8		270 184 B8	È	310 200 C8	Ø	330 216 D8	è	350 232 E8	ø	370 248 F8	8
HTJ	211 137 89		231 153 99	©	251 169 A9	1	271 185 B9	É	311 201 C9	Ù	331 217 D9	é	351 233 E9	ù	371 249 F9	9
VTS	212 138 8A		232 154 9A	a	252 170 AA	o	272 186 BA	Ê	312 202 CA	Ú	332 218 DA	ê	352 234 EA	ú	372 250 FA	10
PLD	213 139 8B	CSI	233 155 9B	≪	253 171 AB	≫	273 187 BB	Ë	313 203 CB	Û	333 219 DB	ë	353 235 EB	û	373 251 FB	11
PLU	214 140 8C	ST	234 156 9C		254 172 AC	¼	274 188 BC	Ì	314 204 CC	Ü	334 220 DC	ì	354 236 EC	ü	374 252 FC	12
RI	215 141 8D	OSC	235 157 9D		255 173 AD	½	275 189 BD	Í	315 205 CD	Ÿ	335 221 DD	í	355 237 ED	ÿ	375 253 FD	13
SS2	216 142 8E	PM	236 158 9E		256 174 AE		276 190 BE	Î	316 206 CE		336 222 DE	î	356 238 EE		376 254 FE	14
SS3	217 143 8F	APC	237 159 9F		257 175 AF	¿	277 191 BF	Ï	317 207 CF	β	337 223 DF	ï	357 239 EF		377 255 FF	15

33	OCTAL
27	DECIMAL
1B	HEX

3.3.2 **ISO Latin-1**

The ISO Latin-1 set has 96 graphic characters. The majority of these are identical to the DEC Supplemental Graphic set, but with a few additional symbols and letters. The ISO Latin-1 set can only be used in VT300 mode.

Table 3-14 ISO Latin-1 Supplemental Character Set

8		9		10		11		12		13		14		15		
	200 128 80	DCS	220 144 90	NBSP	240 160 A0	°	260 176 B0	À	300 192 C0	Ð	320 208 D0	à	340 224 E0	ð	360 240 F0	0
	201 129 81	PU1	221 145 91	¡	241 161 A1	±	261 177 B1	Á	301 193 C1	Ñ	321 209 D1	á	341 225 E1	ñ	361 241 F1	1
	202 130 82	PU2	222 146 92	¢	242 162 A2	2	262 178 B2	Â	302 194 C2	Ò	322 210 D2	â	342 226 E2	ò	362 242 F2	2
	203 131 83	STS	223 147 93	£	243 163 A3	3	263 179 B3	Ã	303 195 C3	Ó	323 211 D3	ã	343 227 E3	ó	363 243 F3	3
IND	204 132 84	CCH	224 148 94	¤	244 164 A4	′	264 180 B4	Ä	304 196 C4	Ô	324 212 D4	ä	344 228 E4	ô	364 244 F4	4
NEL	205 133 85	MW	225 149 95	¥	245 165 A5	µ	265 181 B5	Å	305 197 C5	Õ	325 213 D5	å	345 229 E5	õ	365 245 F5	5
SSA	206 134 86	SPA	226 150 96	¦	246 166 A6	¶	266 182 B6	Æ	306 198 C6	Ö	326 214 D6	æ	346 230 E6	ö	366 246 F6	6
ESA	207 135 87	EPA	227 151 97	§	247 167 A7	·	267 183 B7	Ç	307 199 C7	×	327 215 D7	ç	347 231 E7	÷	367 247 F7	7
HTS	210 136 88		230 152 98	¨	250 168 A8	′	270 184 B8	È	310 200 C8	Ø	330 216 D8	è	350 232 E8	ø	370 248 F8	8
HTJ	211 137 89		231 153 99	©	251 169 A9	1	271 185 B9	É	311 201 C9	Ù	331 217 D9	é	351 233 E9	ù	371 249 F9	9
VTS	212 138 8A		232 154 9A	ª	252 170 AA	º	272 186 BA	Ê	312 202 CA	Ú	332 218 DA	ê	352 234 EA	ú	372 250 FA	10
PLD	213 139 8B	CSI	233 155 9B	«	253 171 AB	»	273 187 BB	Ë	313 203 CB	Û	333 219 DB	ë	353 235 EB	û	373 251 FB	11
PLU	214 140 8C	ST	234 156 9C	¬	254 172 AC	¼	274 188 BC	Ì	314 204 CC	Ü	334 220 DC	ì	354 236 EC	ü	374 252 FC	12
RI	215 141 8D	OSC	235 157 9D		255 173 AD	½	275 189 BD	Í	315 205 CD	Ý	335 221 DD	í	355 237 ED	ý	375 253 FD	13
SS2	216 142 8E	PM	236 158 9E	®	256 174 AE	¾	276 190 BE	Î	316 206 CE	Þ	336 222 DE	î	356 238 EE	þ	376 254 FE	14
SS3	217 143 8F	APC	237 159 9F	¯	257 175 AF	¿	277 191 BF	Ï	317 207 CF	ß	337 223 DF	ï	357 239 EF	ÿ	377 255 FF	15

GR CODES (ISO LATIN-1 SUPPLEMENTAL GRAPHIC)

C1 CODES

33	OCTAL
27	DECIMAL
1B	HEX

3.3.3 DEC Special Graphics

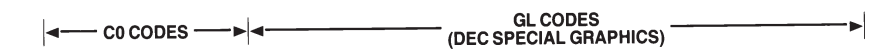
This character set is also called the VT100 Line Drawing character set. It is comprised of ASCII characters and special symbols.

The DEC Special Graphics set can replace either the GL or GR characters. Refer to the Mapping Character Sets topic for more information.

This mapping is compatible with VT100 and VT300 modes.

Table 3-15 DEC Special Graphic Character Set

	0		1		2		3		4		5		6		7	
0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	♦	140 96 60	– SCAN 3	160 112 70
1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	▒	141 97 61	– SCAN 5	161 113 71
2	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	HT	142 98 62	– SCAN 7	162 114 72
3	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	FF	143 99 63	– SCAN 9	163 115 73
4	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	CR	144 100 64	├	164 116 74
5	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	LF	145 101 65	┤	165 117 75
6	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	°	146 102 66	┴	166 118 76
7	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	±	147 103 67	┬	167 119 77
8	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	NL	150 104 68	│	170 120 78
9	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	VT	151 105 69	≤	171 121 79
10	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	┘	152 106 6A	≥	172 122 7A
11	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	┐	153 107 6B	π	173 123 7B
12	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	┌	154 108 6C	≠	174 124 7C
13	CR	15 13 D	GS	35 29 1D	–	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	└	155 109 6D	£	175 125 7D
14	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	┼	156 110 6E	·	176 126 7E
15	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	(BLANK)	137 95 5F	– SCAN 1	157 111 6F	DEL	177 127 7F



33	OCTAL
27	DECIMAL
1B	HEX

3.3.4 National Replacement Character

All National Replacement Character sets are supported. Select the character set with a set mode control sequence.

Table 9-16 NRC Control Sequences

Sequence	Function
$^{C}S_{I}$? 42h	Set National
$^{C}S_{I}$? 42l	Reset National (Set Multinational)

NRC sets are available for the following languages.

Language	NRC Set	Language	NRC Set
United Kingdom	United Kingdom	Italian	Italian
Danish	Norwegian/Danish	Norwegian	Norwegian/Danish
Dutch	Dutch	Portuguese	Portuguese
Finnish	Finnish	Spanish	Spanish
Flemish	French	Swedish	Swedish
French/Belgium	French	Swiss (French)	Swiss
French/Canadian	French Canadian	Swiss (German)	Swiss
German	German		

Each 7-b…acter set. However… replaced character…

Note: NCR s… mode.

Table 9-17 National Replacement Character Sets

Character Set	35	64	91	92	93	94	95	96	123	124	125	126

3.3.5 Character Set Selection

To select a character set, it must first be designated as a G0, G1, G2 or G3 logical set, as in the following sequence:

$^{E}S_{C}$ <Intermediate> <Final>

The intermediate character is selected based on where the set is to be designated (G0, G1, etc.) and whether the set has 94 or 96 characters. 96 character sets cannot be designated as G0.

Table 3-18 Character Set Designation - Intermediate

To Select	Use
94 Character Set	
G0	(
G1	)
G2	*
G3	+

The final character is the designator for the character set.

Table 3-19 Character Set Designation - Final

To Select	Use
ASCII (94)	B
DEC Supplemental Graphic (94)	%5
ISO Latin-1 Supplemental (96)	A
User-preferred Supplemental (94)	<
DEC Special Graphic (94)	0
National Replacement Character Sets (94)	

Example: **E_{SC} +%5**

Designates the DEC Supplemental Graphic set as the G3 logical set.

3.3.6 Mapping Character Sets

Character sets are mapped into use with locking shift and single shift functions. Locking shift functions map a character set into GL or GR where it remains until another locking shift is used.

Table 3-20 Mapping Character Sets with Locking Shifts

Sequence	Function
$^{S}_{I}$	Locking shift 0. Maps G0 into GL
$^{S}_{O}$	Locking shift 1. Maps G1 into GL
$^{E}_{S_C}$ ~	Locking shift 1, right. Maps G1 into GR *
$^{E}_{S_C}$ n	Locking shift 2. Maps G2 into GL *
$^{E}_{S_C}$ }	Locking shift 2, right. Maps G2 into GR *
$^{E}_{S_C}$ o	Locking shift 3. Maps G3 into GL *
$^{E}_{S_C}$ |	Locking shift 3, right. Maps G3 into GR *

* Indicates VT300 mode only.

Single shift functions map the G2 or G3 set into GL for the next character only. After the next character is displayed, the previous character set is restored into GL.

Table 3-21 Mapping Character Sets with Single Shifts

8-Bit Character	7-Bit Equivalent Sequence	Function
$^{S}_{S_2}$	$^{E}_{S_C}$ N	Single shift 2. Maps G2 into GL for the next character.
$^{S}_{S_3}$	$^{E}_{S_C}$ O	Single shift 3. Maps G3 into GL for the next character.

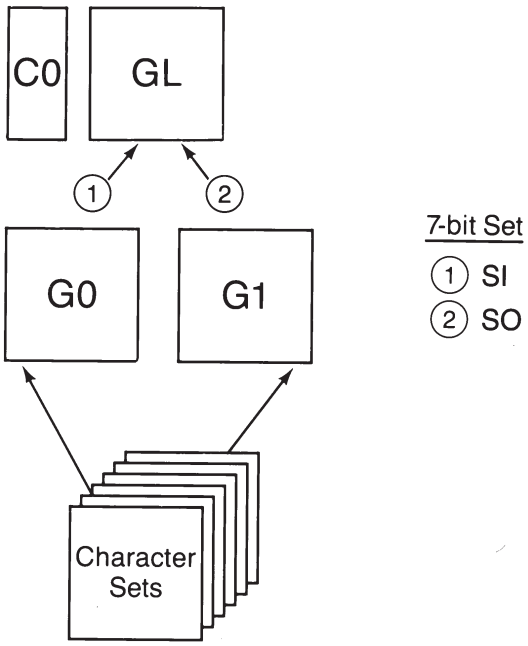


Figure 3-1 Locking Commands (VT100)

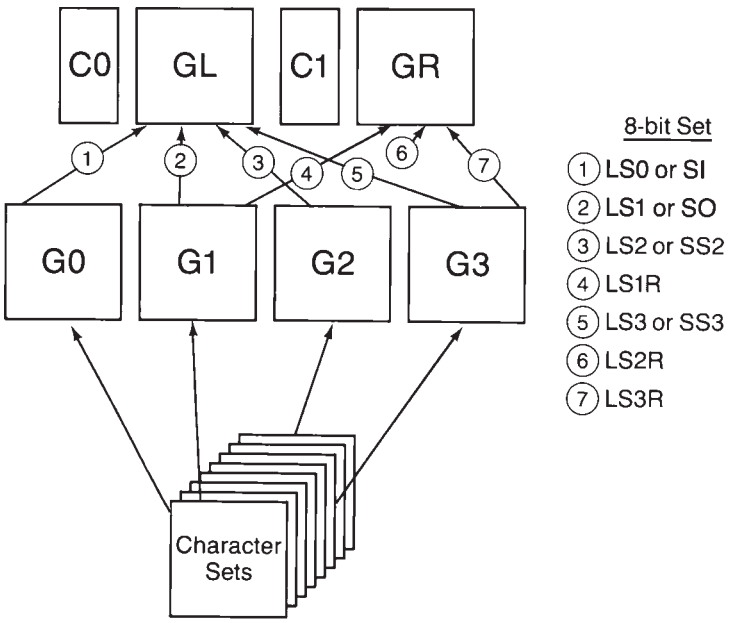


Figure 3-2 Locking and Single Shift Commands

3.4 TRANSMITTED CODES

This section describes the codes that the emulator sends to a program. Key codes generated in VT52 mode are listed if they differ from those in ANSI compatible mode (VT200 and VT100).

3.4.1 Main Keypad

The main keypad consists of standard keys (used to generate letters, numbers, and symbols) and function keys (used to generate special function codes).

3.4.1.1 Standard Keys

The standard keys generate only alphanumeric, ASCII characters. There are no DEC Supplemental characters among the standard keys. However, you can create any DEC Multinational graphics character that is not available through a standard key by typing a valid compose sequence.

Each character is represented by a unique code that is based on the character's position in the code table. Refer to the 7-Bit ASCII Codes table for more information.

3.4.2 Editing Keypad

The codes in the following table are generated by the VT320 editing keypad and cursor keys.

Table 3-22 Codes from Editing Keys

Key	VT320 Mode
Find	$^{C}_{S_I}$ 1 ~
Insert Here	$^{C}_{S_I}$ 2 ~
Remove	$^{C}_{S_I}$ 3 ~
Select	$^{C}_{S_I}$ 4 ~
Prev Screen	$^{C}_{S_I}$ 5 ~
Next Screen	$^{C}_{S_I}$ 6 ~

Table 3-23 Codes from Cursor Control Mode

	Cursor Key Mode		VT52 Mode
Key	Reset Normal	Set Application	Normal and Application
	$^{C}_{S_I}$ A	$^{S}_{S_3}$ A	$^{E}_{S_C}$ A
	$^{C}_{S_I}$ B	$^{S}_{S_3}$ B	$^{E}_{S_C}$ B
	$^{C}_{S_I}$ C	$^{S}_{S_3}$ C	$^{E}_{S_C}$ C
	$^{C}_{S_I}$ D	$^{S}_{S_3}$ D	$^{E}_{S_C}$ D

3.4.3 Auxiliary Keypad

The characters sent by the auxiliary keypad keys depend upon the settings of two features; the operating mode (ANSI or VT52) and the keypad mode (application or numeric).

Table 3-24 Codes from Auxilary Keypad Keys

	VT320/VT100 ANSI Mode		VT52 Mode	
Key	**Numeric**	**Application**	**Numeric**	**Application**
0	0	$^{S}_{S3}$ p	0	$^{E}_{SC}$? p
1	1	$^{S}_{S3}$ q	1	$^{E}_{SC}$? q
2	2	$^{S}_{S3}$ r	2	$^{E}_{SC}$? r
3	3	$^{S}_{S3}$ s	3	$^{E}_{SC}$? s
4	4	$^{S}_{S3}$ t	4	$^{E}_{SC}$? t
5	5	$^{S}_{S3}$ u	5	$^{E}_{SC}$? u
6	6	$^{S}_{S3}$ v	6	$^{E}_{SC}$? v
7	7	$^{S}_{S3}$ w	7	$^{E}_{SC}$? w
8	8	$^{S}_{S3}$ x	8	$^{E}_{SC}$? x
9	9	$^{S}_{S3}$ y	9	$^{E}_{SC}$? z
-	(minus)	$^{S}_{S3}$ m	-	$^{E}_{SC}$? m
,	(comma)	$^{S}_{S3}$ l	,	$^{E}_{SC}$? l
.	(period)	$^{S}_{S3}$ n	.	$^{E}_{SC}$? n
PF1	$^{S}_{S3}$ P	$^{S}_{S3}$ P	$^{E}_{SC}$ P	$^{E}_{SC}$ P
PF2	$^{S}_{S3}$ Q	$^{S}_{S3}$ Q	$^{E}_{SC}$ Q	$^{E}_{SC}$ Q
PF3	$^{S}_{S3}$ R	$^{S}_{S3}$ R	$^{E}_{SC}$ R	$^{E}_{SC}$ R
PF4	$^{S}_{S3}$ S	$^{S}_{S3}$ S	$^{E}_{SC}$ S	$^{E}_{SC}$ S
Enter	$^{C}_{R}$ or $^{C}_{R}{}^{L}_{F}$	$^{S}_{S3}$ M	$^{C}_{R}$ or $^{C}_{R}{}^{L}_{F}$	$^{E}_{SC}$? M

Note: $^{E}_{SC}$ [is the 7-bit equivalent for $^{C}S_{I}$.
$^{E}_{SC}$ O is the 7-bit equivalent for $^{S}_{S3}$.

3.4.4 Top Row Function Keys

On the VT320 keyboard there are 20 top row function keys, F1 through F20. The keys F1 - F5, labeled Hold Screen, Print Screen, Set-Up, Data/Talk, and Break, do not send codes. F6 - F20 send the codes defined below.

Table 3-25 Codes from Top Row Functions

Function Key	Generic Name	VT320 Mode	VT100/VT52 Mode
Hold Screen	F1	*	*
Print Screen	F2	*	*
Set-Up	F3	*	*
Data/Talk	F4	*	*
Break	F5	*	*
F6	F6	C_{SI} 1 7 ~	*
F7	F7	C_{SI} 1 8 ~	*
F8	F8	C_{SI} 1 9 ~	*
F9	F9	C_{SI} 2 0 ~	*
F10	F10	C_{SI} 2 1 ~	*
F11	F11	C_{SI} 2 3 ~	*
F12	F12	C_{SI} 2 4 ~	*
F13	F13	C_{SI} 2 5 ~	*
F14	F14	C_{SI} 2 6 ~	*
Help	(F15)	C_{SI} 2 8 ~	*
Do	(F16)	C_{SI} 2 9 ~	*
F17	F17	C_{SI} 3 1 ~	*
F18	F18	C_{SI} 3 2 ~	*
F19	F19	C_{SI} 3 3 ~	*
F20	F20	C_{SI} 3 4 ~	*

* Indicates that codes are not generated.

3.5.3 Control Characters

Tables 3-31and 3-32 define the action taken by the emulator when it receives C0 and C1 control characters. The VT320 does not recognize all C0 and C1 characters; those not shown in either table are ignored.

Table 3-31 C0 Control Characters

C0	Name	Action
$^{N}U_{L}$	Null	Ignored when received.
$^{E}N_{Q}$	Enquiry	Generates answerback message.
$^{B}E_{L}$	Bell	Generates bell tone.
^{B}S	Backspace	Moves cursor to the left one position.
^{H}T	Horizontal Tabulation	Moves cursor to next tab stop. Does not cause auto wrap.
^{L}F	Line Feed	Causes a line feed.
^{V}T	Vertical Tabulation	Processed as ^{L}F.
^{F}F	Form Feed	Causes a form feed.
^{C}R	Carriage Return	Moves cursor to left margin on current line.
^{S}O (LS1)	Shift Out (Lock Shift G1)	Invokes G1 character set into GL. G1 is designated by a select character set (SCS) sequence.
^{S}I (LS0)	Shift In (Lock Shift G0)	Invokes G0 character set into GL. G0 is designated by a select character set (SCS) sequence.
$^{D}C_{1}$	Device Control 1	Also referred to as Xon. If Xoff support is enabled, $^{D}C_{1}$ clears $^{D}C_{3}$ (Xoff); this causes the emulator to continue sending characters.
$^{D}C_{3}$	Device Control 3	Also referred to as Xoff. If Xoff support is enabled, $^{D}C_{3}$ causes the emulator to stop sending characters until a $^{D}C_{1}$ control character is received.
$^{C}A_{N}$	Cancel	If received during an escape or control sequence, terminates and cancels the sequence. No error character is displayed. If received during a $^{D}C_{S}$, the $^{D}C_{S}$ is terminated and no error character is displayed.
$^{S}U_{B}$	Substitute	If received during an escape or control sequence, terminates and cancels the sequence. Also displays a reverse question mark. If received during a $^{D}C_{S}$, terminates the $^{D}C_{S}$ and displays a reverse question mark.
$^{E}S_{C}$	Escape	Processed as an escape sequence introducer. Terminates any escape control or $^{D}C_{S}$ in progress.
$^{D}E_{L}$	Delete	Ignored when received. Cannot be used as a time fill character.

The equivalent 7-bit code extensions for each 8-bit C1 code are shown in the table below. The code extensions require one more byte than the C1 codes.

Table 3-32 C1 Control Characters

C1	Name	Equivalent 7-Bit	Action
IND	Index	ESC D	Moves cursor down one line in same column.
NEL	Next Line	ESC E	Moves cursor to first position on next line.
HTS	Horizontal Tab Set	ESC H	Sets one horizontal tab stop at column where the cursor is.
RI	Reverse Index	ESC M	Moves cursor up one line in same column.
SS2	Single Shift G2	ESC N	Temporarily invokes G2 character set into GL for the next character. G2 is designated by an SCS sequence.
SS3	Single Shift G3	ESC O	Temporarily invokes G3 character set into GL for the next character. G3 is designated by an SCS sequence.
DCS	Device Control String	ESC P	Processed as opening delimiter of a DCS for device control use.
CSI	Control Sequence Introducer	ESC [	Processed as a control sequence introducer.
ST	String Terminator	ESC \	Processed as a closing delimiter of a string opened by a DCS.

3.5.4 Cursor Positioning

The cursor indicates the active screen position where the next character appears. Cursor positioning can be controlled with the following sequences:

Table 3-33 Cursor Positioning

Name	Sequence	Action
Cursor Up (CUU)	CSI Pn A	Moves cursor up Pn lines in the same column.
Cursor Down (CUD)	CSI Pn B	Moves cursor down Pn lines in the same column.
Cursor Forward (CUF)	CSI Pn C	Moves cursor right Pn columns.
Cursor Backward (CUB)	CSI Pn D	Moves cursor left Pn columns.
Cursor Position (CUP)	CSI Pl;Pc H	Moves cursor to line Pl, column Pc.
Horizontal & Vertical Position (HVP)	CSI Pl;Pc f	Moves cursor to line Pl, column Pc.
Index (IND)	ESC D	Moves cursor down one line in the same column.
Reverse Index (RI)	ESC M	Moves cursor up one line in the same column.
Next Line (NEL)	ESC E	Moves cursor to the first position of the next line.
Save Cursor (DECSC)	ESC 7	The following is saved in terminal memory: – Cursor Position – Graphic Rendition – Character Set Shift State – State of Wrap Flag – State of Origin Mode – State of Selective Erase
Restore Cursor (DECRC)	ESC 8	Restores the states described for DECSC above.

3.5.5 Editing

Editing sequences are used to insert or delete characters and lines at the cursor position.

Table 3-34 Editing

Name	Sequence	Action
Insert Line (IL)	C_{SI} Pn L	Inserts Pn lines at the cursor position.
Delete Line (DL)	C_{SI} Pn M	Deletes Pn lines at the cursor position.
Insert Character (ICH)	C_{SI} Pn @	Inserts Pn blank characters at the cursor position (VT320 mode only).
Delete Character (DCH)	C_{SI} Pn P	Deletes Pn characters starting at the cursor position.

3.5.6 Erasing

The erasing sequences are used to erase characters, lines, etc. from the cursor position.

Table 3-35 Erasing

Name	Sequence	Action
Erase Character (ECH)	C_{SI} Pn X	Erase character at the cursor position and the next Pn-1 characters (VT320 mode only).
Erase In Line (EL)	C_{SI} K	Erase from cursor to end of line, inclusive.
	C_{SI} 0 K	Same as above.
	C_{SI} 1 K	Erase from beginning of line to cursor, inclusive.
	C_{SI} 2 K	Erase the entire line.
Erase In Display (ED)	C_{SI} J	Erase from cursor to end of screen, inclusive.
	C_{SI} 0 J	Same as above.
	C_{SI} 1 J	Erase from beginning of screen to cursor, inclusive.
	C_{SI} 2 J	Erase entire display.
Selective Erase In Line (DECSEL)	C_{SI} **?** K	Erase all erasable characters from cursor to end of line.
	C_{SI} **?** 0 K	Same as above.
	C_{SI} **?** 1 K	Erase all characters from beginning of line to cursor, inclusive.
	C_{SI} **?** 2 K	Erase all erasable characters on the line.
Selective Erase In Display (DECSED)	C_{SI} **?** J	Erase all erasable characters from cursor to end of screen.
	C_{SI} **?** 0 J	Same as above.
	C_{SI} **?** 1 J	Erase all erasable characters from beginning of screen to cursor, inclusive.
	C_{SI} **?** 2 J	Erase all characters in the display.

3.5.7 Line Attributes

Line attributes are display features that affect a complete display line. Select line attributes by using the following sequences:

Table 3-36 Line Attribute Sequences

Sequence	Action
$^{E}S_{C}$ # 3	Double height line, top half
$^{E}S_{C}$ # 4	Double height line, bottom half
$^{E}S_{C}$ # 5	Single width line
$^{E}S_{C}$ # 6	Double width line

3.5.8 Printing

All print operations can be selected using control sequences. But, before you select a print operation, you should check the printer status using the Print Status Report.

Table 3-37 Printer Operations

Operation	Sequence	Action
Auto Print Mode	$^{C}S_{I}$ **? 5 i**	Turns on Auto Print mode. The printed line ends with ^{C}R and the character that moved the cursor off the previous line (^{L}F, ^{F}F, or ^{V}T). Auto Wrap lines end with a line feed.
	$^{C}S_{I}$ **? 4 i**	Turns off Auto Print mode.
Printer Controller Mode	$^{C}S_{I}$ **5 i**	Turns on Printer Controller mode. The terminal sends received characters to the printer without displaying them on the screen.
	$^{C}S_{I}$ **4 i**	Turns off Printer Controller mode.
Print Cursor Line	$^{C}S_{I}$ **? 1 i**	Prints the display line containing the cursor. The Print Cursor Line sequence is complete when the line prints.
Print Screen	$^{C}S_{I}$ **i**	Prints the screen display. The Print Screen sequence is complete when the screen prints.
	$^{C}S_{I}$ **0 i**	Same as above.

3.5.9 Scrolling Region

This sequence is affected by Origin Mode.

Name	Sequence	Action
Set Top & Bottom Margins	$^{C}S_{I}$ **Pt;Pb r**	Pt is the top margin and Pb is the bottom margin. The scrolling region must be at least two lines and Pb must be larger than Pt. The cursor is placed in the home position.

3.5.10 Select C1 Controls

Select C1 Controls can be used to represent C1 control codes in 7-bit or 8-bit form. However, it is recommended that you use DECSCL sequences instead of Select C1 Controls. The advantage is DECSCL performs a soft reset, putting the emulator in a known state, in addition to setting the Terminal mode and the C1 control state.

3.5.10.1 Select 7-bit C1 Transmission (S7C1T)

Name	Sequence	Action
S7C1T	$^{E}s_{C}$ **space F**	Converts all C1 codes returned to the host to their equivalent 7-bit code extensions.

Note: The S7C1T sequence is ignored in VT100 and VT52 modes.

3.5.10.2 Select 8-bit C1 Transmission (S8C1T)

Name	Sequence	Action
S8C1T	$^{E}s_{C}$ **space G**	Returns C1 codes to the application without converting them to their 7-bit code extensions.

3.5.11 Tab Stops

Table 3-38 Tab Stops

Name	Sequence	Action
Set Tab	$^{E}s_{C}$ **H**	Sets a tab stop at the current column.
Clear Tab	$^{C}s_{I}$ **g**	Clears a tab stop at the current column.
	$^{C}s_{I}$ **0 g**	Same as above.
	$^{C}s_{I}$ **3 g**	Clears all tab stops.

3.5.12 Terminal Modes

A mode is a terminal operating state; each mode changes the way the emulator works.

Each mode has an identifying mnemonic name. You can set or reset modes individually or in strings, using set mode (SM) or reset mode (RM) control sequences.

3.5.12.1 Reset Mode (RM)

Resets the ANSI and Digital private modes, individually or in strings.

Mode	Sequence	Action
ANSI	**CSI Ps ;...; Ps l**	Reset sequence for ANSI modes.
DEC Private	**CSI ? ;...; Ps l**	Reset sequence for DEC private modes.

3.5.12.2 Set Mode (SM)

Sets the ANSI and DEC private modes, individually or in strings.

Mode	Sequence	Action
ANSI	**CSI Ps ;...; Ps h**	Set sequence for ANSI mode.
DEC Private	**CSI ? ;...; Ps h**	Set sequence for DEC Private mode.

Table 3-39 Selectable Modes Summary

Name	Code	Set Mode	Reset Mode
ANSI/VT52	DECANM	N/A	VT52
Auto Repeat	DECARM	On	**CSI ? 2 l** Off
Auto Wrap	DECAWM	**CSI ? 8 h** On	**CSI ? 8 l** Off
Backarrow Key	DECBKM	**CSI ? 7 h** BS	**CSI ? 7 l** DEL
Character Set	DECNRCM	**CSI ? 67 h** National	**CSI ? 67 l** Multinational
Column	DECCOLM	**CSI ? 42 h** 132 Column	**CSI ? 42 l** 80 Column
Cursor Key	DECCKM	**CSI ? 3 h** Application	**CSI ? 3 l** Cursor
Insert/Replace	IRM	**CSI ? 1 h** Insert	**CSI ? 1 l** Replace
Keyboard Action	KAM	**CSI 4 h** Locked	**CSI 4 l** Unlocked
Keypad	DECKPAM/ DECKPNM	**CSI 2 h** Application	**CSI 2 l**
Line Feed/New Line	LNM	**ESC =** New Line	Line Feed
Numeric Keypad	DECNKM	**CSI 20 h** Application	**CSI 20 l** Numeric
		CSI ? 66 h	**CSI ? 66 l**

Table 3-39 Selectable Modes Summary (cont'd)

Name	Code	Set Mode	Reset Mode
Origin	DECOM	Origin **$^{C}s_{I}$? 6 h**	Absolute **$^{C}s_{I}$? 6 l**
Print Extent	DECPEX	Full Screen **$^{C}s_{I}$? 19 h**	Scroll Rgn **$^{C}s_{I}$? 19 l**
Print Form Feed	DECPFF	On **$^{C}s_{I}$? 18 h**	Off **$^{C}s_{I}$? 18 l**
Screen	DECSCNM	Reverse **$^{C}s_{I}$? 5 h**	Normal **$^{C}s_{I}$? 5 l**
Scrolling	DECSCLM	Smooth **$^{C}s_{I}$? 4 h**	Jump **$^{C}s_{I}$? 4 l**
Select Status Display	DECSASD	**$^{C}s_{I}$ Ps$}** Ps=0 main display Ps=1 status line	
Select Status Line Type	DECSSDT	**$^{C}s_{I}$ Ps $~** Ps=0 none Ps=1 indicator Ps=2 host-writable	
Send/Receive	SRM	Off **$^{C}s_{I}$ 12 h**	On **$^{C}s_{I}$ 12 l**

3.5.12.3 ANSI/VT52 Mode (DECANM)

In ANSI mode, reset selects VT52 compatibility mode. In VT52 mode, the emulator responds to Digital private sequences like a VT52 terminal.

Mode	Sequence	Action
Reset	**$^{C}s_{I}$? 2 l**	Sets the emulator to VT52 mode.

Note: There is no Set mode for ANSI/VT52 mode.

3.5.12.4 Auto Repeat Mode (DECARM)

Specifies whether or not keys automatically repeat their character when held down.

Mode	Sequence	Action
Set	**$^{C}s_{I}$? 8 h**	Keys autorepeat when pressed for more than 0.5 seconds
Reset	**$^{C}s_{I}$? 8 l**	Keys do not auto

3.5.12.5 Auto Wrap Mode (DECAWM)

Selects where received characters appear when the cursor is at the right margin.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 7 h**	Selects auto wrap. Characters received when the cursor is at the right margin appear on the next line at the left margin.
Reset	$^{C}S_{I}$ **? 7 l**	Turns off auto wrap. Characters received when the cursor is at the right margin are overwritten.

3.5.12.6 Backarrow Key Mode (DECBKM)

Selects whether the emulator sends a delete or backspace for the backarrow key.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 67 h**	Move cursor one position to the left (backspace).
Reset	$^{C}S_{I}$ **? 67 l**	Delete previous character.

3.5.12.7 Character Set Mode (DECNRCM)

Determines whether the emulator uses NRCs or the DEC multinational character set.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 4 2 h**	Select National mode. Generates 7-bit characters from NRC sets.
Reset	$^{C}S_{I}$ **? 4 2 l**	Selects Multinational mode. Generates 8-bit characters from the multinational character set, including 7-bit characters from the ASCII set.

3.5.12.8 Column Mode (DECCOLM)

Column mode selects the number of columns per line; 80 or 132.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 3 h**	Selects 132 columns.
Reset	$^{C}S_{I}$ **? 3 l**	Selects 80 columns.

3.5.12.9 Cursor Key Mode (DECCKM)

Cursor Key mode determines the character sent by the cursor keys.

Mode	Sequence	Action
Set	$^{C}S_{I}$? 1 h	Causes the cursor keys to send application control functions.
Reset	$^{C}S_{I}$? 1 l	Causes the cursor keys to send ANSI cursor control sequences.

3.5.12.10 Insert/Replace Mode (IRM)

Insert/Replace mode determines how the emulator adds characters to the screen.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **4 h**	Selects Insert mode. New characters move old characters to the right.
Reset	$^{C}S_{I}$ **4 l**	Selects Replace mode. New characters replace old characters at the cursor position. The old character is erased.

3.5.12.11 Keyboard Action Mode (KAM)

Keyboard Action mode lets your program lock and unlock the keyboard. When the keyboard is locked it cannot send codes to the program.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **2 h**	Locks the keyboard.
Reset	$^{C}S_{I}$ **2 l**	Unlock the keyboard, unless it is locked by $^{D}C_{3}$.

3.5.12.12 Keypad Mode (DECKPAM/DECKPNM)

The auxiliary keypad generates either numeric characters or control functions.

Mode	Sequence	Action
Application (DECKPAM)	$^{E}S_{C}$ **=**	Selects Application keypad mode. Keypad keys send application control functions.
Numeric (DECKPNM)	$^{E}S_{C}$ **>**	Selects Numeric keypad mode. Keypad keys send numeric, comma, period, and minus sign codes. PF1 - PF4 send control functions.

3.5.12.13 Line Feed/New Line Mode (LNM)

Line Feed/New Line mode selects the control character(s) sent to the application by the Return and Enter keys.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **2 0 h**	Causes a received ^{L}F, ^{F}F, or ^{V}T code to move the cursor to the first column of the next line. Return sends ^{C}R and ^{L}F.
Reset	$^{C}S_{I}$ **2 0 l**	Causes a received ^{L}F, ^{F}F, or ^{V}T code to move the cursor to the next line in the current column. Return sends ^{C}R only.

3.5.12.14 Numeric Keypad Mode (DECNKM)

Numeric Keypad mode selects whether the emulator sends numeric characters or application sequences for the numeric keypad.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 66 h**	Numeric keypad sends application sequences.
Reset	$^{C}S_{I}$ **? 66 l**	Numeric keypad sends numeric characters.

3.5.12.15 Origin Mode (DECOM)

Origin mode allows cursor addressing relative to a user-defined origin.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 6 h**	Selects home position as the top margin of the user-defined scrolling region. The cursor cannot move out of the scrolling region. All cursor positioning is relative to the top of the scrolling region.
Reset	$^{C}S_{I}$ **2 0 l**	Causes a received ^{L}F, ^{F}F, or ^{V}T code to move the cursor to the next line in the current column. Return sends ^{C}R only.

3.5.12.16 Print Extent Mode (DECPEX)

Print Extent mode selects the full screen or the scrolling region for a print screen operation.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 1 9 h**	Selects full screen for a print screen operation.
Reset	$^{C}S_{I}$ **? 1 9 l**	Selects the scrolling region for a print screen operation.

3.5.12.17 Print Form Feed Mode (DECPFF)

This mode determines whether the emulator sends a print termination character after a screen print. The form feed character (^{F}F) serves as the print termination character.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 1 8 h**	Selects ^{F}F as the print termination character. The emulator sends this character to the printer after each print screen operation.
Reset	$^{C}S_{I}$ **? 1 8 l**	Selects no termination character. The emulator does not send a ^{F}F to the printer after each print screen operation.

3.5.12.18 Screen Mode (DECSCNM)

Screen mode selects a normal or reverse video display on the screen.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 5 h**	Select reverse video.
Reset	$^{C}S_{I}$ **? 5 l**	Select normal screen.

3.5.12.19 Scrolling Mode (DECSCLM)

There are two methods of scrolling; jump and smooth scroll.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 4 h**	Select smooth scroll.
Reset	$^{C}S_{I}$ **? 4 l**	Select jump scroll.

3.5.12.20 Select Status Display (DECSASD)

Selects whether the emulator sends data to the main display (first 24 lines) or the status line (25th line). Available in VT300 mode only.

$^{C}S_{I}$ **Ps $** }

Ps	Display option
0	data is sent to the main display only
1	data is sent to the status line only

3.5.12.21 Select Status Line Type (DECSSDT)

Enables the host to select the type of status line.

$^{C}S_{I}$ Ps $ ~	Ps	Status line selection
	0	no status line
	1	indicator
	2	host-writable

Note: If the status line is changed from indicator to host-writable, the new status line is empty.

When the host-writable status line is selected, most control functions affecting the main display affect the status line. The following table lists the exceptions.

Table 3-40 Control Function Effects on the Status Line

Function	Effect
ANSI mode	Ignored if received in the status line.
C1 transmissions	Affects main display and status line.
Cursor position controls	Affects only the column parameters.
Hard terminal reset	Erases and exits status line.
Insert/replace mode	Affects main display and status line.
Screen alignment test	No effect.
Screen mode	Affects main display and status line.
Scrolling mode	Affects main display and status line.
Select character set	The same character set is used in both the main display and status line.
Set conformance test	Exits status line.
Soft terminal reset	Exits status line.
Tab stops	Affects main display and status line.
Text cursor enable mode	The cursor can be individually enabled in the main display or status line.

3.5.12.22 Send/Receive Mode (SRM)

Send/Receive mode turns local echo on or off.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **1 2 h**	Disables local echo. When the emulator sends characters to the host, the host must echo characters back to the emulator.
Reset	$^{C}S_{I}$ **1 2 l**	Enables local echo. When the emulator sends characters, the characters are automatically sent to the screen.

3.5.12.23 Text Cursor Enable Mode (DECTCEM)

Text Cursor Enable mode determines if the text cursor is visible.

Mode	Sequence	Action
Set	$^{C}S_{I}$ **? 2 5 h**	Makes the cursor visible.
Reset	$^{C}S_{I}$ **? 2 5 l**	Make the cursor invisible.

3.5.13 Terminal Reset Mode

There are two terminal reset control sequences: a soft terminal reset, and a hard terminal reset.

3.5.13.1 Soft Terminal Reset

The DECSTR sequence sets the terminal to the states listed below. The DECSTR sequence is as follows:

$^{C}S_{I}$ **! p**

Table 3-41 Soft Terminal Reset States

Sequence	State
Text Cursor	On
Insert/Replace	Replace
Origin Mode	Absolute
Auto Wrap	Off
Keyboard Action	Unlocked
Keypad Mode	Numeric
Cursor Key Mode	Normal
Top Margin	1
Bottom Margin	24
Character Sets	VT320 defaults
Cursor Position	Home
SGR Write State	Normal
Origin Mode	Normal (reset)
National/Multinational	Multinational
Video Character Attributes	Normal
Selective Erase Attributes	Normal (erasable)

3.5.13.2 Hard Terminal Reset

A hard terminal reset is implemented by clicking ***Execute - Reset***.

3.5.14 Programming User Defined Keys (UDKs)

When the terminal is in VT300 mode, you can download key sequences into the programmable function keys using DECUDK device control strings. To access the keys programmed value, press Shift and the function key.

The emulator has 512 bytes available for 20 programmable function keys. (The VT320 only has 256 bytes available for 15 function keys). Space is supplied on a first come-first serve basis. After the 512 bytes are used, you must clear space to redefine keys. There are three ways to clear space:

1) Redefine a key (or keys) using a DECUDK.
2) Clear a key (or keys) using a DECUDK.
3) Clear the definition by clicking ***Execute - Reset.***

3.5.14.1 DECUDK DCS Format

The Device Control String (DCS) format for downloading UDKs is as follows:

$^{D}_{CS}$ Pc;Pl | Ky1/st1;ky2/st2;...kyn/stn $^{S}_{T}$

$^{D}_{CS}$ The Device Control String introducer, $^{D}_{CS}$ is an 8-bit character. $^{E}_{SC}$ P is the 7-bit coding equivalent.

Pc The Pc (clear parameter) determines which keys are cleared, and when. A value of 0 (or no value) clears all keys, and 1 clears each key to be reloaded just before reloading it.

Pl The Pl (lock parameter) determines whether the key definitions are locked or not after you load them. A value of 0 (no value) locks the keys (non-define). A value of 1 does not lock them (define).

| This is the final character. It designates the control string as a DECUDK.

Kyn/stn This is the key definition string. Each string consists of a key selector number (Kyn) and a string parameter (stn) separated by a slash. The Kyn specifies the key to be redefined and the stn is the encoded contents of the string. The stn consists of hex pairs.

$^{S}_{T}$ The string terminator is an 8-bit control character that is expressed as $^{E}_{SC}$ \ for 7-bit coding.

The following is a list of definable keys and their identifying values:

Token	Value	Token	Value	Token	Value
UDK1	12	UDK11	23	UDK15	28
UDK2	13	UDK12	24	UDK16	29
UDK3	14	UDK13	25	UDK17	31
UDK4	15	UDK14	26	UDK18	32
UDK5	16	UDK8	19	UDK19	33
UDK6	17	UDK9	20	UDK20	34
UDK7	18	UDK10	21		

The tokens **UDK1 - UDK5** are not assigned in the default keyboard configuration. They must be assigned with the Keyboard Mapping feature.

3.5.14.2 Guidelines for Loading Keys

- Use the UDK clear parameter to reclaim key definition space.
- Generally, you should not leave keys unlocked.
- The host must keep track of the available space for definitions.
- If you redefine a key, the old sequence is lost.
- The emulator uses a special lock for the programmable keys. The lock can be turned on with a DECUDK, but can only be unlocked by the UDK unlock parameter. The lock acts globally over all programmable keys.
- All key definitions are stored in volatile RAM. If there is a power loss, the key definitions are lost. An invalid $^{D}C_{S}$ in a key definition causes an aborted load. An aborted load locks the keys, saves the successfully loaded keys, and sends the rest of the DECUDK sequence to the screen.

3.5.14.3 Examples for Using DECUDK

Example 1: **$^{D}C_{S}$ 0;1| ^{S}T**

Clears all of the UDKs.

Example 2: **$^{D}C_{S}$ 1;0| ^{S}T**

Locks the UDKs.

Example 3: **$^{D}C_{S}$ 1;1|34/5052494E54 ^{S}T**

Clears and leaves **UDK20** unlocked. Then, defines **UDK20** as “PRINT”.

P = 50 hex
R = 52 hex
I = 49 hex
N = 4E hex
T = 54 hex

Note: $^{D}C_{S}$ is also represented by the 7-bit equivalent of $^{E}S_{C}$ P.
^{S}T is also represented by the 7-bit equivalent of $^{E}S_{C}$ \ .

3.5.15 DCS Private Control Sequences

DCS private sequences are control sequences supported only by a Minisoft emulator. They are not available on VT320 terminals.

Table 3-42 DCS Private Control Sequences

Name	Sequence	Action
Enable User Status Line	$^{C}S_{I}$ **0;0|**	Enables the emulator status line for the host. When the emulator receives the enable command, it displays the previous user-defined status line. If the status line was not previously downloaded, it is cleared.
Disable User Status Line	$^{C}S_{I}$ **0;1|**	Disables the display of the user-defined status line and redisplays the emulator status line. The contents of the downloaded status line are not destroyed. It may be redisplayed by sending an enable sequence.
Erase Status Line	$^{C}S_{I}$ **0;2|**	Erases the status line and clears the down-loaded data.
Write Status Line	$^{C}S_{I}$ **0;3;Pc| ...string...** ^{S}T	Writes the characters between the vertical bar and the string terminator to the status line starting at column Pc.
Set/Reset Local Echo	$^{C}S_{I}$ **2;n|**	Enables local echo if n=1. Disables local echo if n=0.
WordPerfect Mode	$^{C}S_{I}$ **3;n|**	Enables WordPerfect mode if n=1, disables if n=0.
Printer Port Control	$^{C}S_{I}$ **4;pl|**	Controls the assignment of the printer port. Where **pl** is: 0=none, 1=LPT1, 2=LPT2, 3=LPT3, 4=COM1, 5=COM2, 6=COM3
Execute Emulator Command	$^{C}S_{I}$ **5| ...command string...**^{S}T	Sends the command string to the emulator for execution. The command string can contain any valid emulator command or reference a command file.
Request Product Identification	$^{C}S_{I}$ **6|**	If this sequence is sent to a Minisoft emulator, the emulator returns an identification report to the host in the format: $^{E}S_{C}$n xxxx Where: **n** is a single ASCII digit indicating the number of characters to follow (n not included). **xxxx** is the product identification string
Set Lines Per Screen	$^{C}S_{I}$ **7;pl|**	Where: **pl** is the number of lines per screen.

Note: $^{C}S_{I}$ (Hex 9B) is the C1 Control Sequence Introducer. The 7-bit equivalent of $^{C}S_{I}$ is $^{E}S_{C}$ [.
^{S}T (Hex 9C) is the C1 String Terminator. The 7-bit equivalent of ^{S}T is $^{E}S_{C}$ \.

3.5.15.1 Example - DCS Private Sequence

$^{C}S_{I}$ 7m $^{C}S_{I}$ 0;3;1|User Defined Status Line ^{S}T

Writes “User Defined Status Line” in reverse video. The status line must have been previously enabled.

$^{C}S_{I}$ 6|

Sent by the host to the emulator, generates the following identification report:

$^{E}S_{C}$ 5 ETERM32

3.6 REPORTS

Reports are sent by the emulator in response to requests from the host computer. These new reports provide device attributes, operating status and terminal state and mode information to the host. The host uses the reports to match the computing environment and emulator.

3.6.1 Device Attributes

Device attributes are used to give the host information regarding the emulator.

3.6.1.1 Primary Device Attributes

Primary device attributes include the service class code and basic attributes. The response of the emulator to this request depends on the type of terminal selected for emulation in ***Setup - Terminal***.

Table 3-43 Primary Device Attribtes

Exchange	Sequence
Host to Emulator	CS_I c or CS_I 0 c
Emulator to Host	CS_I ? Psc ; Ps1 ; ... Psn c
	Psc Service class code based on operating level
	1 level 1 (VT100)
	6 level 1 (VT102)
	62 level 2 (VT200)
	63 level 3 (VT300)
	Ps1 Basic attributes supported by the emulator
	1 132 columns
	2 printer port
	6 selective erase
	7 soft character set
	8 user-defined keys
	9 national replacement character sets

3.6.1.2 Secondary Device Attributes

The secondary device attributes include identification code, firmware version and hardware options.

Table 3-44 Secondary Device Attributes

Exchange	Sequence
Host to Emulator	**CSI > c** or **CSI > 0 c**
Emulator to Host	**CSI > Pp ; Pv ; Po c** **Pp** Emulator identification code 24 VT320 **Pv** Firmware version level of the emulator **Po** Hardware options 0 there are no options for the VT320

3.6.2 Device Status Reports

The emulator uses device status reports to give the host information on cursor position, keyboard dialect, operating status, printer status and user-defined keys.

3.6.2.1 Cursor Position

Exchange	Sequence	Function
Host to Emulator	**CSI 6 n**	Host requests cursor position
Emulator to Host	**CSI Pl ; Pc R**	The emulator specifies Pl (line) and Pc (column) as current cursor position

3.6.2.2 Keyboard Dialect

Exchange	Sequence	Function
Host to Emulator	**CSI ? 26 n**	Host requests keyboard
Emulator to Host	**CSI ? 27 ; Pd n** **Pd** Keyboard dialect 1 North American	Keyboard dialect (Pd) is reported

3.6.2.3 Operating Status

Exchange	Sequence		Function
Host to Emulator	$^{C}s_{I}$ **5 n**		Host requests the emulator's operating status
Emulator to Host	$^{C}s_{I}$ **0 n**	or	The emulator indicates there is **no** malfunction
	$^{C}s_{I}$ **3 n**		The emulator indicates there is a malfunction

3.6.2.4 Printer Status

Exchange	Sequence	Function
Host to Emulator	$^{C}s_{I}$ **? 15 n**	Host requests current printer status
Emulator to Host	$^{C}s_{I}$ **? 13 n**	No printer
	$^{C}s_{I}$ **? 10 n**	Printer ready
	$^{C}s_{I}$ **? 11 n**	Printer not ready

3.6.2.5 User-Defined Key (UDK) Status

This control function is only valid in VT300 mode.

Exchange	Sequence	Function
Host to Emulator	$^{C}s_{I}$ **? 25 n**	Host requests if UDKs are locked or unlocked
Emulator to Host	$^{C}s_{I}$ **? 20 n**	UDKs are unlocked
	$^{C}s_{I}$ **? 21 n**	UDKs are locked

3.6.3 Terminal State Reports

Terminal state reports include the current setting for all of the emulator's features except user-defined keys. The host can use the report information to save the current state. The host can then temporarily change the operating state and, later, restore the emulator to the saved state. This control function is valid only in VT300 mode.

Table 3-45 Terminal State Report

Exchange	Sequence	
Host to Emulator	$^{C}S_{I}$ Ps $ u	
	Ps	Report requested
	0	ignored, no report sent
	1	terminal state report requested
Emulator to Host	$^{D}C_{S}$ 1 $ s D1...Dnn <checksum1><checksum2> ^{S}T	
	D1...Dnn	Data string indicating the status of emulator functions. There are nn bytes in the string. D1...Dnn are each in the range of column 4 rows 0 to 15 in the code table. Bit 6 of each Dn byte is always on; bit 7 is always off.
	<checksum1>	2-byte checksum of all data (D1...Dnn) in the report. Checksum is equal to the
	<checksum2>	2's complement of the sum of all data elements in the report (D1+D2+...Dn).

Note: Software should not expect the format of the terminal state report to be the same for all VT300 terminals.

3.6.3.1 Restore Terminal State

This sequence is sent from the host to restore the emulator to the previous state specified in the terminal state report.

Table 3-46 Restore Terminal State

Restore	Sequence	
Host to Emulator	$^{D}C_{S}$ Ps $ p D...D ^{S}T	
	Ps	Indicates whether or not the host succeeds in restoring the terminal state. Must be 1 for a successful restore.
	0	error, restore ignored
	1	restore to previous terminal state based on terminal state report
	D...D	Data string containing the restored information. This string is identical to the data string used by the terminal state report.

Note: If an invalid value is received, no changes are made.

3.6.4 Presentation State Reports

There are two presentation reports: cursor information and tab stop. The host can use the report information to save the current state. The host can then temporarily change the presentation state and, later, restore the emulator to the saved state. This control function is only valid in VT300 mode.

3.6.4.1 Request Presentation State Report

Table 3-47 Request Presentation State Report

Request	Sequence
Host to Emulator	C_{S_I} Ps $ w
	Ps Indicates which report is requested
	0 error, request ignored
	1 cursor information report
	2 tab stop report

3.6.4.2 Cursor Information

The cursor information report gives the status of the cursor position, including visual attributes and character protection attributes.

Table 3-48 Cursor Information Report

Report	Sequence
Emulator to Host	D_{C_S} **1 $ u D...D** S_T
	D...D Data string of cursor information in the following format: Pr; Pc; Pp; Srend; Satt; Sflag; Pgl; Pgr; Scss; Sdesig

The individual parameters that make up the data string are described in the following table.

Table 3-49 Cursor Information Report Data String

Parameter	Description
Pr	Row number of the cursor position
Pc	Column number of the cursor position
Pp	Current page number - always 1 for VT320
Srend	One or more characters indicating visual attributes currently in use for writing. The character converts to an 8-bit binary number. The attributes can then be found in the following list. The list is ordered from most significant bit (8) to least significant bit (1).

Bit	Attribute	Bit Value
8		Always 0 (off)
7		Always 1 (on)
6	Extension indicator	0 no more attribute data 1 another character of visual attribute data follows this one
5		Always 0 (off)
4	Reverse video	0 off 1 on
3	Blinking	0 off 1 on
2	Underline	0 off 1 on
1	Bold	0 off 1 on

Satt	One or more characters indicating selective erase attributes currently set for writing. The character converts to an 8-bit binary number. The attributes can be found in the following list.

Bit	Attribute	Bit Value
8		Always 0 (off)
7		Always 1 (on)
6	Extension indicator	0 no more protection data 1 another character of selective erase data follows this one
5		0 - Reserved for future use
4		0 - Reserved for future use
3		0 - Reserved for future use
2		0 - Reserved for future use
1	Selective erase	0 off 1 on

Table 3-49 Cursor Information Report Data String (cont'd)

Parameter	Description
Sflag	Character(s) indicating flags and modes the terminal must save. The character converts to an 8-bit binary number. **Bit** **Attribute** **Bit Value** 8 Always 0 (off) 7 Always 1 (on) 6 Extension indicator 0 no more flag data 1 another character of flag data follows this one 5 0 Reserved for future use 4 Autowrap 0 autowrap not pending 1 autowrap pending 3 Single shift 3 setting 0 single shift 3 is off 1 G3 is mapped into GL for the next typed character only 2 Single shift 2 setting 0 single shift 2 is off 1 G2 is mapped into GL for the next typed character only 1 Origin Mode 0 origin mode reset 1 origin mode set
Pgl	Indicates the number of the logical character set (G0 through G3) mapped into GL. 0 G0 is in GL 1 G1 is in GL 2 G2 is in GL 3 G3 is in GL
Pgr	Indicates the number of the logical character set (G0 through G3) mapped into GR. 0 G0 is in GR 1 G1 is in GR 2 G2 is in GR 3 G3 is in GR
Scss	Indicates the size of the character sets in G0 - G3. The character converts to an 8-bit binary number. **Bit** **Attribute** **Bit Value** 8 Always 0 (off) 7 Always 1 (on) 6 Extension indicator 0 no more size data 1 another character of character size data follows this one 5 0 Reserved for future use 4 G3 set size 0 94 characters 1 96 characters 3 G2 set size 0 94 characters 1 96 characters 2 G1 set size 0 94 characters 1 96 characters 1 G0 set size 0 94 characters 1 96 characters
Sdesig	String of intermediate and final characters indicating the character sets designated as G0 through G3. These final characters are the same as those used in select character set sequences.

3.6.4.3 Tab Stop Report

If the presentation state report requests information on the tab stops, the emulator returns the following:

Table 3-50 Tab Stop Report

Report	Sequence	
Emulator to Host	$^{D}C_{S}$ **2 $ u D...D** $^{S}_{T}$	
	D...D	Data string indicating the column number location of each tab stop. Column numbers are separated by slashes (/).

3.6.4.4 Restore Presentation State

The restore presentation state report restores the emulator to a previous saved state based on one of the presentation state reports: cursor information or tab stop. The information from only one report at a time can be restored. This sequence is only valid in VT300 mode.

Table 3-51 Restore Presentation State

Restore	Sequence	
Host to Emulator	$^{D}C_{S}$ **Ps $ t D...D** $^{S}_{T}$	
	Ps	Indicates the format of the data string, D...D, which corresponds to one of the presentation state report formats
	0	error, restore ignored
	1	cursor information report format
	2	tab stop report format
	D...D	Data string containing the restored information. This string is identical to the one used in the report.

Note: If there is an invalid value in the restore sequence, the rest of the sequence will be ignored. This may leave the emulator in a partially restored state.

3.6.5 Mode Settings

The host can request current settings of any ANSI or DEC private modes. The emulator returns a report indicating which modes are set and reset. The host uses the report information to save the current mode settings. The host then temporarily changes the modes and, later, restores the emulator to the saved modes with the set and reset mode sequences. This control function is only valid in VT300 mode.

3.6.5.1 Request Mode

The host sends the following sequence to find out if a particular mode is set or reset. There is a different sequence for ANSI and DEC private modes.

Table 3-52 Request ANSI Mode

Request	Sequence	
Host to Emulator	C_{S_I} **Pa $ p**	
	Pa	Indicates the ANSI mode on which the host is requesting information.

The ANSI modes (Pa) are listed in the following table.

Table 3-53 ANSI Modes

Pa	Mode
2	Keyboard action
3	Control representation *
4	Insert/replace
10	Horizontal editing
12	Send/receive
20	Line feed/new line

* Control representation is not supported.

Note: Control representation and horizontal editing are permanently reset.

Table 3-54 Request DEC Private Mode

Request	Sequence	
Host to Emulator	C_{S_I} **? Pd $ p**	
	Pd	Indicates the DEC private mode on which the host is requesting information

The DEC private modes (Pd) are listed in the following table.

Table 3-55 DEC Private Modes

Pd	Mode
1	Cursor keys
2	ANSI
3	Column
4	Scrolling
5	Screen
6	Origin
7	Autowrap
8	Autorepeat
18	Print form feed
19	Printer extent
25	Text cursor enable
42	NRC set
66	Numeric keypad
67	Backarrow key
68	Keyboard usage *

3.6.5.2 Report Mode

The ANSI mode and DEC private mode reports are given in the following table. The emulator can report on only one mode at a time.

Table 3-56 ANSI Mode and DEC Private Mode Report

Report	Sequence
Emulator to Host	$^{C}s_{I}$ Pa ; Ps $ y
(ANSI mode)	**Pa** Indicates reported ANSI mode (see Table 3-53)
	Ps Indicates mode setting
	0 mode not recognized
	1 set
	2 reset
	3 permanently set
	4 permanently reset
Emulator to Host	$^{C}s_{I}$? Pd ; Ps $ y
(DEC private mode)	**Pd** Indicates reported DEC private mode (see Table 3-55)
	Ps Indicates mode setting
	0 mode not recognized
	1 set
	2 reset
	3 permanently set
	4 permanently reset

3.6.5.3 Set Mode

There is a separate set sequence for the ANSI modes and DEC private modes. Some of these may be affected by soft or hard terminal resets.

Table 3-57 ANSI and DEC Private Mode Set Sequence

Set Mode	Sequence	
Host to Emulator	$^{C}S_{I}$ Pa ; ... ; Pa h	
(ANSI form)	Pa	Indicates the ANSI mode to set. See Table 3-53 for the list of ANSI modes. More than one value can be used in the sequence.
Host to Emulator	$^{C}S_{I}$? Pd ; ... ; Pd h	
(DEC private form)	Pd	Indicates the DEC private mode to set. See Table 3-55 for the list of DEC private modes. More than one Pd value can be used in the sequence.

3.6.5.4 Reset Mode

There is a separate reset sequence for the ANSI modes and DEC private modes. Some of these may be affected by soft or hard terminal resets.

Table 3-58 ANSI and DEC Private Mode Reset Sequence

Reset Mode	Sequence	
Host to Emulator	$^{C}S_{I}$ Pa ; ... ; Pa l	
(ANSI form)	Pa	Indicates the ANSI mode to reset. See Table 3-53 (ANSI Modes). More than one value can be used in the sequence.
Host to Emulator	$^{C}S_{I}$? Pd ; ... ; Pd l	
(DEC private form)	Pd	Indicates the DEC private mode to reset. See Table 3-55 (DEC Private Modes). More than one Pd value can be used in the sequence.

3.6.6 Save and Restore Cursor State

The save cursor sequence stores many of the emulator's selections and settings. The host can then temporarily change the settings. The restore cursor sequence restores the emulator to the saved settings.

Table 3-59 Saving and Restoring the Cursor State

Name	Sequence	Function
Save cursor	$^{E}S_{C}$ **7**	Saves the following: - Cursor position - Character attributes set by select graphic rendition sequence - Character set (G0, G1, G2, G3) currently in GL or GR - Wrap flag (autowrap or no autowrap) - State of origin mode - Selective erase attribute - Any single shift 2 or single shift 3 functions sent
Restore cursor	$^{E}S_{C}$ **8**	Restores the emulator to the saved state. If nothing was saved with the save cursor sequence, the following occurs: - Moves cursor to home position (upper left of screen) - Resets origin mode - Turns all character attributes off - Maps ASCII set into GL, and DEC Supplemental Graphic set into GR

Note: The emulator maintains a separate save cursor buffer for the main display and the status line. A separate operating state for the main display and the status line can be saved.

3.6.7 Control Function Settings

The host can request the current selection or setting of the following control functions: active status display, conformance level, status line type, top and bottom margins and graphic rendition.

The emulator returns a report with the requested information. The host can use the report information to save the current setting. The host can then temporarily change the control function settings and later, restore the emulator to the saved settings. This control function is only valid in VT300 mode.

Note: The control function request can only ask about one function at a time.

Table 3-60 Control Functions Setting Report

Exchange	Sequence	
Host to Emulator	**$^{D}C_S$ $ q D...D $^{S}_T$**	
	D...D	Indicates the control function in question. Consists of the intermediate and/or final characters of the control function requested.
	$}	active status display
	“q	character attribute
	”p	conformance level
	$~	status line type
	r	top and bottom margins
	m	graphic rendition
Emulator to Host	**$^{D}C_S$ Ps $ r D...D $^{S}_T$**	
	Ps	Indicates if a request from the host is valid
	0	invalid
	1	valid
	D...D	Indicates the current setting of the control function requested. Consists of all control function characters except the $^{C}S_I$ or $^{E}S_C$[introducer characters.

3.6.8 User-Preferred Supplemental Set

The host can request the current user-preferred supplemental character set. This control function is only valid in VT300 mode.

Table 3-61 User-Preferred Supplemental Character Set

Exchange	Sequence	Function
Host to Emulator	**$^{C}S_I$ & u**	Requests current user-preferred supplemental set
Emulator to Host	**$^{D}C_S$ 0 ! u % 5 $^{S}_T$**	DEC Supplemental Graphic set
	$^{D}C_S$ 1 ! u A $^{S}_T$	ISO Latin-1 Supplemental set

WYSE, SCO-ANSI & ANSI PROGRAMMING

OVERVIEW

This chapter describes the character encoding concepts for the WYSE, SCO-ANSI and ANSI terminals. It covers control functions, such as control characters and escape sequences. Control functions are used in a program to specify how the emulator processes, sends and displays characters. Each control function has a unique name and each name has a unique, mnemonic abbreviation.

4.1 WYSE PROGRAMMING SEQUENCES

4.1.0.1 Screen Feature Codes

Table 4-1 Screen Feature Codes

n	Screen Cursor	Setting
0	Cursor display off	
1	Cursor display on	(default)
2	Steady block cursor	
3	Blinking line cursor	
4	Steady line cursor	
5	Blinking block cursor	(default)
A	Normal protect character	
6	Reverse protect character	
7	Dim protect character	
8	Screen display off	
9	Screen display on	(default)
:	80-column screen	(default)
;	132-column screen	
<	Smooth scroll at 1 row per second	
=	Smooth scroll at 2 rows per second	
>	Smooth scroll at 4 rows per second	
?	Smooth scroll at 8 rows per second	
@	Jump scroll	(default)

4.1.1 Cursor Addressing

The following instructions apply only to cursor addressing in the WY-50; TVI-910/920/925.

1) Send **E_{SC}=rc** to move the cursor to a specific row and column of an 80-column screen.

   Where: **r** is row code (see Row/Column Codes).

   c is column code (see Row/Column Codes).

2) Send **E_{SC} a rr R ccc C** to move the cursor to a specified row and column of either an 80- or 132-column screen.

   Where: **rr** is ASCII encoded decimal value of row relative to home, one or two digits.
   R is ASCII R.
   ccc is ASCII encoded decimal value of column relative to home, up to three digits.
   C is ASCII C.

Example: **E_{SC} a 1 R 1 C** positions the cursor at true home.
E_{SC} a 10 R 10 C positions the cursor at row 10, column 10.

4.1.2 Row/Column Codes

Table 4-2 Row/Column Codes

Row	WY-50 TVI-910/920/925 Row Code	Column	WY-50 TVI-910/920/925 Column Code	Column (cont'd)	WY-50 TVI-910/920/925 Column Code (cont'd)
1	(space)	1	(space	41	H
2	!	2	!	42	I
3	“	3	”	43	J
4	#	4	#	44	K
5	$	5	$	45	L
6	%	6	%	46	M
7	&	7	&	47	N
8	‘	8	‘	48	O
9	(	9	(	49	P
10	)	10	)	50	Q
11	*	11	*	51	R
12	+	12	+	52	S
13	‘	13	“	53	T
14	-	14	-	54	U
15	.	15	.	55	V
16	/	16	/	56	W
17	0	17	0	57	X
18	1	18	1	58	Y
19	2	19	2	59	Z
20	3	20	3	60	[
21	4	21	4	61	\
22	5	22	5	62	]
23	6	23	6	63	^
24	7	24	7	64	_
		25	8	65	‘
		26	9	66	a
		27	:	67	b
		28	;	68	c
		29	<	69	d
		30	=	70	e
		31	>	71	f
		32	?	72	g
		33	@	73	h
		34	A	74	i
		35	B	75	j
		36	C	76	k
		37	D	77	l
		38	E	78	m
		39	F	79	n
		40	G	80	o

4.1.3 Display Attributes

An attribute is written into the current cursor location and occupies a space.

Send ESC A n ATTR to set a display attribute for a special WY-50 message field, and send ESC G ATTR to set a display attribute for individual data.

Where: **N** is the display field code (see below).
ATTR is the attribute code (see Attribute Codes)

Table 4-3 Display Field Codes

WY-50 Display Field	n
Application Display Area	0
Function Key Labeling Line	1
Local Message Field	2
Host Message Field	3

Table 4-4 Display Attribute Codes

Display Attributes	ATTR
Blank	1
Blink	2
Dim	p
Normal	0
Reverse	4
Underscore	8

4.1.4 Attribute Codes

The complete attribute codes are listed in the following table.

Table 4-1 Attribute Codes

ATTR	Display Attributes
(space)	Space code (20H)
0	Normal
1	Blank (no display)
2	Blink
3	Blank
4	Reverse
5	Reverse and blank
6	Reverse and blink
7	Reverse, blink and blank
8	Underscore
9	Underscore and blank
:	Underscore and blink
;	Underscore, blink and blank
<	Underscore and reverse
=	Underscore, reverse and blank
>	Underscore, reverse and blink
?	Underscore, reverse, blink and blank
p	Dim
q	Dim and blank
r	Dim and blink
s	Dim, blink and blank
t	Dim and reverse
u	Dim, reverse and blank
v	Dim, reverse and blink
w	Dim, reverse, blink and blank
x	Dim and underscore
y	Dim, underscore and blank
z	Dim, underscore and blink
{	Dim, underscore, blink and blank
|	Dim, underscore and reverse
}	Dim, underscore, reverse and blank
~	Dim, underscore, reverse and blink

4.1.5 Control Codes

Press CTRL with the control key (see table below) to enter the control code via the keyboard.

Where: **Control key** is the associated alphanumeric key (below).

Table 4-1 Control Codes

Control Code	ASCII Hex Code	Display Symbol	Control Key	WY-50 Action
NULL	00	(blank)	@ or ‘	No action.
SOH	01	S_H	A or a	No action.
STX	02	S_X	B or b	No action.
ETX	03	E_X	C or c	No action.
EOT	04	E_T	D or d	No action.
ENQ	05	E_Q	E or e	Returns ACK if not busy.
ACK	06	A_K	F or f	No action.
BEL	07	B_L	G or g	Sounds the beeper.
BS	08	B_S	H or h	Backspaces the cursor.
HT	09	H_T	I or i	Tabs the cursor.
LF	0A	L_F	J or j	Moves cursor down.
VT	0B	V_T	K or k	Moves cursor up.
FF	0C	F_F	L or l	Moves cursor right.
CR	0D	C_R	M or m	Moves cursor to far left position of row.
SO	0E	S_O	N or n	Unlocks the keyboard.
SI	0F	S_I	O or o	Locks the keyboard.
DLE	10	*	P or p	No action.
DC1 (XON)	11	*	Q or q	Enables the transmitter
DC2	12	*	R or r	Turns on auxilary print; data displays.
DC3 (XOFF)	13	*	S or s	Stops transmission to host computer.
DC4	14	*	T or t	Turns off auxilary and transparent print.
NAK	15	*	U or u	No action.
SYN	16	*	V or v	No action.
ETB	17	*	W or w	No action.
CAN	18	*	X or x	Transparent print.
EM	19	*	Y or y	No action.
SUB	1A	*	Z or z	Clears all unprotected characters to spaces.
ESC	1B	*	{ or [	Initiates an escape sequence.
FS	1C	*	| or \	No action.
GS	1D	*	} or]	No action.
RS	1E	*	^ or ~	Moves cursor to the home position.
US	1F	*	_ or DEL	Moves cursor down one row to far left position.

* Due to font limitations, we are unable to display the actual graphic display symbol at this time.

4.1.6 Escape Codes

Table 4-2 Escape Codes

Escape Code	Action
ESC (space)	Reports the terminal identification to the host computer. Sends 50 CR.
ESC !	Writes all unprotected character positions with a specified attribute code. This has a format of: ESC ! ATTR (Where: ATTR is the atrtribute code. See Attribute Codes.)
ESC “	Unlocks the keyboard.
ESC #	Locks the keyboard.
ESC &	Turns the protect submode on and prevents the auto scroll operation.
ESC ‘	Turns the protect submode off and allows the auto scroll operation.
ESC (	Turns the write protect submode off.
ESC)	Turns the write protect submode on.
ESC *	Clears the screen to nulls. The protect submode is turned off.
ESC +	Clears the screen to spaces. The protect submode is turned off.
ESC , Clears the screen to protected spaces. The protect submode is turned off.	
ESC -	Moves the cursor to a specified text segment. This has a multiple code sequence of: ESC - arc (Where: n is the text segment number - 0 or 1; r is the row code - see Row/Colum Codes; and c is the column code - see Row/Column Codes.
ESC .	Clears all unprotected character positions with a specified character code. This has a format of: ESC. CODE (Where: CODEis the character hex value.)
ESC \	Transmits the active text segment number and cursor address.
ESC 0	Clears all tab settings.
ESC 1	Sets a tab stop.
ESC 2	Clears a tab stop.
ESC 4	Sends all unprotected characters from the start-of-row to the host computer.
ESC 5	Sends all unprotected characters from the start-of-text to the host computer.
ESC 6	Sends all characters from the start-of-row to the host computer.
ESC 7	Sends all characters from the start-of-text to the host computer.
ESC 8	Enters a start-of-message character (STX).
ESC 9	Enters an end-of-message character (ETX).
ESC :	Clears all unprotected characters to nulls.

Table 4-7 Escape Codes (cont'd)

Escape Code	Action
ESC ?	Transmits the cursor address for the active text segment of an 80-column screen only. The format is: rc CR (Where: r is the row code - see Row/Column Codes; and c is the column - see Row/Column Codes.)
ESC @	Sends all unprotected characters from the start-of-text to the auxiliary port. Each row is terminated wit: CR LF NULL
ESC A	Sets a video attribute for a specific message field or the entire application display area. This has a multiple code sequence of: ESC A n ATTR (Where: n os the fireld code - See Display Attributes; and ATTR is the attribute code - see Attribute Codes.)
ESC B	Places the terminal in block mode.
ESC C	Places the terminal in a conversation mode.
ESC D	Selected the full duplex or half duplex conversation modes. This has the multiple code sequence: ESC D x)Where: x is F for full duplex mode or H for half duplex mode.)
ESC E	Inserts a row of spaces.
ESC F	Enters a message in the host message field. This has a format of: ESC F aaaa CR (Where: aaaa is a character string of up to 46 characters for an 80-column mode screen or up to 100 characters for a 132-column screen.
ESC G	Sets a video attribute within the application display are. The attribute occupies a space. This has a multiple code sequence of: ESC G ATTR (Where: ATTR is the attribute code - see Attribute Codes.)
ESC H	Enters a graphic character at the cursor location. This has a multiple code sequence of: ESC H x (Where: x is the graphic chracter code - see Graphic Character.) ESC H STX (CTRL B) turns on the graphic submode. ESC H ETX (CTRL C) turns off the graphic submode.
ESC I	Moves the cursor left to the previous tab stop.
ESC J	Activates the alternate text segment.
ESC K	Activates the alternate text segment. See ESC J.
ESC L	Sends all characters unformatted to the auxiliary port. Attribute codes are sent as spaces. Row-end sequences are not sent.
ESC M	Causes the terminal to send the character at the cursor postions to the host computer.
ESC N	Turns the no scoll submode on.
ESC O	Turns the no scoll submode off.
ESC P	Sends all protected and unprotected characters to the auxiliary port, regardless of the mode setting.
ESC Q	Inserts a character.
ESC R	Deletes a row.
ESC S	Sends a message unprotected.
ESC T	Erases all characters/
ESC U	Turns the monitor submode on.
ESC V	Sets a protected column.
ESC W	Deletes a character.
ESC X	Turns the monitor submode off.
ESC Y	Erases all characters to the end of the active text segment and replaces them with spaces.
ESC]	Activates text segment 0.

Table 4-7 Escape Codes (cont'd)

Escape Code	Action
ESC ‘	Sets the screen features. This has the following multiple code sequence: ESC ‘ n (Where: n is the screen feature code - See Screen Feature Codes.)
ESC a	Moves the cursor to a specified row and column for an 80-column or a 132-column scree. This has a format of: ESC a rr R ccc C (Where: rr is the ASCII encoded decimal value of the row; R is the ASCII R; ccc is the ASCII encoded decimal value of the column; and C is the ASCII C. For example: ESC a 1 R 1 C postions the cursor at true home.
ESC b	Transmits the cursor address to the host computer for the active text segment. This format is: rr R ccc C (Where: rr is the ASCII encoded decimal value of the row; ; R is the ASCII R; ccc is the ASCII encoded decimal value of the column; and C is the ASCII C.
ESC i	Moves the cursor to the next tab stop on the right.
ESC j	Moves the cursor up one row and begins srolling at top row.
ESC k	Turns the local edit submode on.
ESC l	Turns the duplex edit submode on.
ESC p	Sends all character unformatted to the auxiliary port. Attribute codes are sent as spaces. Row-end sequences are not sent. The action is the same as ESC L.
ESC q	Turns the insert submode on.
ESC r	Turns the insert submode off.
ESC s	Sends a message.
ESC t	Erases all characters from the current cursor location to the end of the row and replaces them with nulls.
ESC u	Turns the monitor submode off. See ESC X.
ESC x	Changes the screen display format. The sequences are: ESC x 0 for a full screen; 24 rows by 80 or 132 columns. ESC x 1 HSR for a horizontal screen position. (Where: HSR is the row code for the row number 2 to 24 on which the lower text segment starts (see Row/Column Codes).
ESC y	Erases all characters from the current cursor location to the end of the active text segment and replaces them with nulls.
ESC z	Enters a message into a selected function key label field or programs a user-defined sequence for a function key (maximum of eight label fields, shiftable to 16 for an 80-column screen; maximum of 16 label fields, shiftable to 32 for a 132-column screen). The message format is: ESC z n aaaa CR (Where: n is the field code - see Function Key Field Codes/Default Value Codes; aaa is a character string of up to eight characters for an 80-column screen or up to seven character for a 132-column screen. ESC z n CR clears a particular function key label field. ESC z DEL turns off the shifted function key labeling line. The function key program format is: ESC z value SEQ DEL (Where: value is the default value code - see Function Key Field Codes/Default Value Codes; SEQ is the program sequence up to eight bytes (256 byte maximum for all function keys.)
ESC {	Moves the cursor to the home position of the text segment.
ESC }	Activates text segment 1.

4.1.7 Function Value Field Codes/Default Value Codes

Table 4-1 Field Codes and Default Value Codes

Function Key	Field Code	Default Value Code
F1	0	@
Shift F1	P	‘
F2	1	A
Shift F2	Q	a
F3	2	B
Shift F3	R	b
F4	3	C
Shift F4	S	c
F5	4	D
Shift F5	T	d
F6	5	E
Shift F6	U	e
F7	6	F
Shift F7	V	f
F8	7	G
Shift F8	W	g
F9	8	H
Shift F9	X	h
F10	9	I
Shift F10	Y	i
F11	:	J
Shift F11	Z	j
F12	;	K
Shift F12	[	k
F13	<	L
Shift F13	\	l
F14	=	M
Shift F14	]	m
F15	>	N
Shift F15	*	n
F16	?	O
Shift F16	_	o

Note: Field codes (unshifted message) and (shifted message) specify the entire function keys labeling line as one message field of up to 78 characters for an 80-column screen or up to 130 characters for a 132-column screen.
80-column screen = eight function key label fields, shiftable to 16.
132-column screen = sixteen function key label fields, shiftable to 32.

4.1.8 Key Tokens

Only those keys which generate codes in conversation mode are listed in the table below. Unless noted, the shifted key positions generate the same code as the unshifted position. All alphanumeric keys generate the standard ASCII codes.

Table 4-2 Key Tokens

Command Keys	Generated Code
PRINT	ESC P
SEND	ESC 7
Cursor Position Keys	
Cursor Down	CTRL J (OAH)
Cursor Left	CTRL H (08h0
Cursor Right	CTRL L (0CH)
Cursor Up	CTRL K (08h)
Home	CTRL ^ (1EH)
Shift Home	ESC {
PAGE NEXT	ESC K
PAGE PREV	ESC J
Editing Keys	
CLR LINE	ESC T
CLR SCRN	ESC Y
DEL CHAR	ESC W
DEL LINE	ESC R
INS	ESC q
INS CHAR	ESC Q
INS LINE	ESC E
REPL	ESC r

4.2 SCO-ANSI PROGRAMMING SEQUENCES

4.2.1 Character Attributes

Table 4-3 Character Attributes

Sequence	Function
$^{C}S_{I}$ 0 m	Reset attributes
$^{C}S_{I}$ 1 m	Bold on
$^{C}S_{I}$ 4 m	Underline on
$^{C}S_{I}$ 5 m	Blink on
$^{C}S_{I}$ 7 m	Reverse video on
$^{C}S_{I}$ = Pn E	Set or clear the blink bit (Pn = 0 or 1). Same as $^{C}S_{I}$ 5 m *
$^{C}S_{I}$ = Cn F	Set normal foreground color to Cn *
$^{C}S_{I}$ = Cn G	Set normal background color to Cn *
$^{C}S_{I}$ = Cn H	Set reverse foreground color to Cn - Ignored *
$^{C}S_{I}$ = Cn I	Set reverse background color to Cn - Ignored *

* Specific to SCO 2.3 and above (Non-ANSI)

4.2.2 Character Sets

Table 4-2 Character Sets

Sequence	Function
$^{C}S_{I}$ 10 m	Select primary font. Causes 8-bit PC character set to be used as the font. PC characters in the control character range are not displayed.
$^{C}S_{I}$ 11 m	Select first alternate font. Same as above except all characters other than the Escape character are displayed.
$^{C}S_{I}$ 12 m	Select second alternate font. Displays PC character set in the 80h and above range to be displayed as the lower character set.
$^{C}S_{I}$ Pn g	Display the character from cell Pn.

4.2.3 Color Attributes

4.2.3.1 ANSI Color Attributes

Table 4-3 ANSI ISO Color Sequences

Sequence	Function
$^{C}S_{I}$ 3 Pc m	Set foreground color from ISO color table
$^{C}S_{I}$ 4 Pc m	Set background color from ISO color table
$^{C}S_{I}$ 8 m	Set blank - invisible characters

Table 4-4 ANSI ISO Color Table

Pc	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan

4.2.3.2 SCO Xenix Color Attributes

Table 4-5 SCO Xenix Color Sequences

Sequence	Function
$^{C}S_{I}$ 2 ; Pf ; Pb m	Set foreground (Pf) and background colors (Pb)

Table 4-6 SCO Xenix Color

Cn	Color	Cn	Color
0	Black	8	Dark Grey
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	Light Grey	15	White

4.2.4 Columns

Table 4-7 Columns

Sequence	Function
$^{C}S_{I}$? 3 l	Set 80 columns

4.2.5 Cursor Positioning

Table 4-8 Cursor Positioning

Sequence	Function
$^{C}S_{I}$ Pn Z	Move cursor backwards Pn stops
$^{C}S_{I}$ Pn S	Scroll up Pn lines
$^{C}S_{I}$ Pn T	Scroll down Pn lines
$^{C}S_{I}$ P1; P2 H	Direct cursor position
$^{C}S_{I}$ P1; P2 f	Same as above
$^{C}S_{I}$ Pn A	Move cursor up Pn lines
$^{C}S_{I}$ Pn B	Move cursor down Pn lines
$^{C}S_{I}$ Pn C	Move cursor right Pn columns
$^{C}S_{I}$ Pn D	Move cursor left Pn columns
$^{C}S_{I}$ Pn ‘	Position cursor to column Pn
$^{C}S_{I}$ Pn a	Move cursor Pn positions to the right relative. Does not wrap.
$^{C}S_{I}$ Pn d	Move cursor to row Pn
$^{C}S_{I}$ Pn e	Move cursor down Pn rows
$^{C}S_{I}$ Pn F	Move cursor to beginning of line Pn lines up
$^{C}S_{I}$ Pn E	Move cursor to beginning of line Pn lines down

4.2.6 **Inserting**

Table 4-9 Inserting

Sequence	Function
$^{C}S_{I}$ Pn J	Erase display
Pn = 0	From cursor to end of display
Pn = 1	From cursor to beginning of display
Pn = 2	Entire display
$^{C}S_{I}$ Pn K	Erase in line
Pn = 0	From cursor to end of line
Pn = 1	From beginning of line to cursor
Pn = 2	Entire line
$^{C}S_{I}$ Pn X	Erase Pn number of characters

4.2.7 Key Assignments

SCO ANSI uses function keys F1-F12, Ctrl F1-F12, Shift F1-F12, and Ctrl-Shift F1-F12. Although the emulator has tokens for these keys, they are not currently available through the Keyboard Mapper.

Table 4-10 Key Assignments

Key	Code
F1 -F12	ESC [M...ESC [V+
Shift F1-F12	ESC [Y...ESC [Z ESC [a...ESC [j
Ctrl F1-F12	ESC [k...ESC [v
Ctrl-Shift F1-F12	ESC [w...ESC [z ESC [@...ESC [}
Up Arrow	ESC [A
Dn Arrow	ESC [B
Right Arrow	ESC [C
Left Arrow	ESC [D
Keypad 0-9	0...9 (NUMLCK in numeric mode)
Home	ESC [H
PgUp	ESC [I
End	ESC [F
PgDn	ESC [G
Ins	ESC [L
Del	0x7F
Shift Tab	ESC [Z
Ctrl Enter	0x81
Ctrl Home	0x82
Ctrl PgUp	0x83
Ctrl BS	0x84
Ctrl End	0x85
Ctrl PgDn	0x86
Ctrl KP -	0x87
Ctrl KP +	0x88
Ctrl Left Arrow	0x89
Ctrl Right Arrow	0x8a

4.2.8 Keyboard Control

Table 4-11 Keyboard Control

Sequence	Function
$^{C}S_{I}$ 2 h	Lock keyboard
$^{C}S_{I}$ 2 l	Unlock keyboard

4.2.9 Report

Table 4-12 Report

Sequence	Function
C_{SI} 2 i	Send screen to host with a line feed after each line.

4.3 ANSI PROGRAMMING SEQUENCES

4.3.1 ANSI Color Support

ANSI color support allows the character, character cell, and screen background colors to be selected directly by sending control sequences from the host.

ANSI colors are selected through extensions to the VT320 Set Character Attribute control sequence. The following table describes the control sequences supported.

Table 4-13 Character and ANSI Color Attributes

Escape Sequence	Function
Set Character Attributes and ANSI Colors	
$^{C}S_{I}$Ps;Ps;...m	
	Character attributes
Ps = 0	Resets all colors and video attributes to defaults
Ps = 1	Bold on. If the text color has been changed using an ANSI color control sequence, bold will be the intensified text color. Otherwise, bolded text will display as configured in the ***Setup - Terminal - Display - Color Setup...***.
Ps = 4	Underscore on. Always uses the colors selected in the ***Setup - Terminal - Display - Color Setup...***.
Ps = 5	Blink on
Ps = 7	Reverse video on. Always uses the colors selected in the ***Setup - Terminal - Display - Color Setup...***.
Ps = 22	Bold off, normal intensity
Ps = 24	Underscore off
Ps = 25	Blink off
Ps = 27	Reverse video off, positive image
	Character Colors (low intensity unless bolded)
Ps = 30	Black
Ps = 31	Red
Ps = 32	Green
Ps = 33	Yellow (displays as brown unless bolded)
Ps = 34	Blue
Ps = 35	Magenta
Ps = 36	Cyan
Ps = 37	White
Ps = 39	White

Table 4-12 Character and ANSI Color Attributes (cont'd)

Escape Sequence	Function
Set Character Attributes and ANSI Colors	
$^{C}_{S_I}$Ps;Ps;...m	
	Character Cell Color (always low intensity colors)
Ps = 40	Sets the cell color to the current background color
Ps = 41	Red
Ps = 42	Green
Ps = 43	Yellow (displays as brown)
Ps = 44	Blue
Ps = 45	Magenta
Ps = 46	Cyan
Ps = 47	White
Ps = 49	Sets the cell color to the current background color
	Direct Index Control Using a Prefix
< index	Specifies the character color index
= index	Specifies the character cell color index
> index	Specifies the screen background index
ANSI Color Indexes	
0 = black	
1 = red	
2 = green	
3 = yellow	
4 = blue	
5 = magenta	
6 = cyan	
7 = white	

The following examples use the emulator command DISPLAY to locally test the character attributes and colors.

Example 1: CMD>**DISPLAY"$^{C}_{S_I}$1;35mBOLD magenta characters$^{C}_{S_I}$ 0m"** or
CMD>**DISPLAY"$^{C}_{S_I}$1;mBOLD magenta characters$^{C}_{S_I}$0m"**

Displays the character string in bold magenta characters at the current cursor position.

Example 2: CMD>**DISPLAY"$^{C}_{S_I}$5;7mReverse blink characters$^{C}_{S_I}$25m"**

Displays the character string using the blink attribute, 5, and the reverse video attribute, 7. After the characters display, the blink is turned off, 25. Subsequent characters display in reverse video.

Example 3: CMD>**DISPLAY"$^{C}_{S_I}$1;33;43mYellow chars/brown cell $^{C}_{S_I}$0m"**

Displays the character string using the bold attribute and character color 33 to give yellow characters. The character cell color, 43, shows as brown directly around each character.

DYNAMIC DATA EXCHANGE

OVERVIEW

Dynamic Data Exchange (DDE) is a method of exchanging information between two independent Windows applications. These applications carry on a “conversation” by posting messages to each other. The application that initiated the “conversation” is the “client”, and the responding program is the “server”.

The emulator can be used as a client, a server, or both. When used as the client, the emulator provides a complete set of DDE commands for interacting with any DDE server. In addition, as a server, the emulator supports symbol linking and remote command execution. All DDE commands are part of the Emulator Command Language (ECL). These commands can be run from a script file, the command line, or the DDE Command Builder dialog box.

Several Windows applications currently support DDE. Consult your program’s documentation for more information.

5.1 USING DDE

Data exchanges that do not require ongoing interaction from the operator can be fully automated with DDE. The emulator establishes a link to another application for the sole purpose of exchanging data — after which the emulator and the other application can exchange data without operator involvement.

DDE can be used in commands for the following purposes:

- Start another application.
- Send data to another application.
- Get data from another application.
- Carry out commands in another application.

You can also implement a broad range of local and host application features including:

- Establishing a link to real-time host data, then transferring the information locally to your PC immediately upon change.
- Performing data queries between applications, such as a spreadsheet querying the host for current numbers from its database.
- Creating a compound document, i.e., a Word file with a graphics chart produced by a graphics program, in which the information for the graphics program comes directly from the host. Using DDE, the chart will be updated upon change of the host data, without changing the rest of the document.

When exiting a copy of the emulator any associated links and any client or server DDE conversations are closed.

5.1.1 DDE Concepts

DDE utilizes some unique terminology which is important to understand before using DDE.

Client vs. Server

The client initiates a conversation with a server, or sends commands to the server to execute. Both the client and the server can terminate the conversation.

Conversations and Transactions

When two applications exchange DDE messages, they are engaged in a conversation. The messages that are passed back and forth are transactions.

The emulator can be engaged in several conversations at the same time, acting as the client in some and as the server in others. These conversations can be between the same application, or different applications. In addition, these conversations may be with other instances of the emulator.

DDE transactions can be one-time data transfers, continuous “links” in which applications send updates to each other as data changes, or commands that are executed by the receiving program. Not all DDE servers allow execution of commands. Consult your DDE program’s documentation.

5.1.2 Service Names, Topic Names, and Item Names

Before initiating a conversation, both applications must agree upon the service, topic, and item names. The DDE syntax of the client application determines how the emulator server recognizes these names.

Service Name

Each DDE conversation is identified by the service name (formerly known as application name) and topic; the client and server agree upon this before the conversation is initiated. The default can be overridden using either the Server Name option (in the DDE dialog box) or the command SET DDE-SERVERNAME. This can be used when multiple copies of the emulator are running simultaneously, and client applications need to distinguish between them in order to talk to the window running on the desired host with the appropriate settings.

Topic Name

The DDE topic is the way data is classified so that multiple data items can be exchanged during a conversation. The topic is typically a filename for those applications operating on file-based documents. Other applications use an application-specific name. Topic and data items are used when a client application begins a DDE conversation with the emulator as the DDE server. Supported topcs include “System”, “ECL”, and “Settings”.

Item Name

All requests must reference an item name which matches a client request to the proper server response. The data item values can be passed from the server to the client and vice versa.

5.1.3 Server Topics

DDE clients can address the emulator as a server during a conversation. Topics and data items are used when a client application starts a DDE conversation with the emulator; the way these are compiled into actual DDE commands is determined by the DDE syntax of the client’s application. The emulator supports the following topics:

System	Provides information to the client about what topics, items, and data formats the server supports. In addition, the System topic can be used to retrieve the server’s current status.
ECL	Allows the client to retrieve data from variables within the emulator and execute ECL commands.
Settings	Provides information about the current settings of the emulator.

5.2 SYSTEM TOPIC

Permits a DDE client to ask a server, such as the emulator, which topic names, item names, and data formats it supports. It also provides general information about the application’s DDE support and accesses the emulator‘s DDE server status.

System topic items are accessed with DDE data requests. Each request returns a specific type data.

To find out which servers are present and the kinds of information they can provide, a client can request a conversation on the System topic with the service name set to NULL (“”).

5.2.1 System Topic Items

Contained within the System topic are pre-defined items that provide specific information. The emulator supports the following system topic items:

SysItems Returns a tab-separated list of items supported under the System topic by this server (SysItems, Topics, Format, Status and StatusNum).

Topics Returns a tab-separated list of topics supported by the emulator DDE server. The topics currently supported are: System, ECL, and Settings.

Format Returns a tab-separated list of clipboard formats supported by the emulator DDE server. Currently, the only format supported is "TEXT".

Status Returns a status string that describes the status of the prior DDE server operation. The string's format is as follows:

"Status n : status description"

Where: **N** is a numeric status code.

A DDE client can use data requests (or establish a permanent link) to monitor the Status item, and receive continuous reports of the server's status. A second conversation can be maintained by the client for this purpose. This information is essential for a client application that runs complex ECL scripts using the DDE execute message (see Executing ECL Commands).

5.3 ECL TOPIC

The ECL (Emulator Command Language) topic allows access to EM320's command language when the emulator is acting as a DDE server. This allows development of sophisticated systems of execution control and dynamic data exchange between other applications and the emulator (and hence host computers and networks).

5.3.1 ECL Topic Items

The Emulator Command Language (ECL) allows the use of symbols (also known as variables) to hold data values. All command language symbols (variables) are valid ECL topic data items. The following sections discuss the various actions that can be performed using the ECL topic from a client application. These include:

- Requesting the value of an ECL variable - Global symbols only
- Changing the value of an ECL variable
- Creating an Advise Data Link to an ECL variable
- Executing ECL commands or command file

5.3.2 Requesting the Value of an ECL Variable

DDE request messages can be issued from a client application to obtain the value of any emulator command language variable. Even though the item requested may be a numeric symbol, all data items sent to the client are in text format. The client application must convert this text value to numeric if necessary.

The following example connects to another instance of the emulator and requests the value of the variable COUNT.

Example: **DDE CONNECT "ms320"**
"ECL" CONVS = **DDE REQUEST 'CONV' "COUNT"**
RESULT DDE DISCONNECT 'CONV'

The value of COUNT in the server instance of emulator is placed in the variable RESULT.

Note: This example assumes that the global variable COUNT exists in the server instance of the emulator. If the global symbol is not found or not initialized, the value returned from the DDE REQUEST will be zero.

5.3.3 Changing the Value of an ECL Variable

DDE poke messages can be issued from a client application to change the value of any emulator variable. All data items sent to the emulator must be in text format. For numeric variables, the value is translated by the emulator automatically.

The following example connects to another instance of the emulator and sets the value of the variable COUNT.

Example: **DDE CONNECT "ms320"**
"ECL" CONV
DDE POKE 'CONV' "COUNT" "200" DDE DISCONNECT 'CONV'

The value of COUNT in the server instance of the emulator would be set to 200.

Note: If the global variable COUNT does not already exist in the server instance of the emulator, it is created and assigned the passed value.

5.3.4 Creating an Advise Data Link to an ECL Variable

DDE advise messages can be issued from a client application to create and Advise Data Link to an ECL variable. Whenever the value of the ECL variable changes, the client application is automatically notified and the new value is sent. As with the DDE request messages, all data items sent to the client are in text format.

Example: You can update the value of a variable that changes frequently because of the host connection, into your Excel spreadsheet. Create an Advise Data Link from Excel to the emulator symbol. Enter the following DDE link into the desired cell of the spreadsheet :

=ms320|ECL!HOSTDATA

This command uses the service name "MS320", the topic "ECL", and links the spreadsheet cell to an ECL variable called HOSTDATA. Whenever the value of HOSTDATA changes, Excel is automatically updated with the new value.

5.3.5 Executing ECL Commands or Command Files

The DDE execute message allows the client to send commands to the emulator server for execution. The following examples illustrate the execute process.

Example 1: **"CLS"**
"STR1 := Dialing..."

Example 2: **"@LOGIN"**

Execute the command file LOGIN.ECF.

5.3.6 Settings Topic

The Settings topic provides query access to a limited number of settings within the emulator. Valid data item names in this topic include the emulator command language SET parameters. Requesting the value of a Settings parameter returns a text string containing the current value of that setting. The Settings topic supports DDE REQUEST only (DDE ADVISE or DDE POKE are not supported).

The data items currently supported include:

- SERVERNAME
- TERMINAL /WIDTH
- TERMINAL /LINES

Example: **DDE CONNECT "ms320"**
"SETTINGS" CONV
DDE REQUEST 'CONV' "TERMINAL /LINES"
RESULT DDE DISCONNECT CONV

The variable RESULT would contain the current number of display lines for the server instance of the emulator.

5.4 DDE COMMANDS

DDE commands appear in uppercase letters (e.g., DDE CONNECT). The standard syntax is:

DDE CONNECT "service name" "topic name" variable

Refer to Chapter 7 (command Language) for more information.

Note: When entering DDE commands from the DDE> prompt, do not preceed the command with DDE.

5.4.1 DDE Server Operation

The emulator can also be used as a server that allows command execution, data retrieval, and data updates.

ECL commands used to change server operations can be entered at the emulator command line prompt or set in the DDE Setup dialog box.

Refer to Chapter 7 (Command Language) for the SET DDE... commands.

5.4.2 DDE Error Facility

Whenever a DDE command is completed, the emulator sets a status condition code in the symbol $STATUS to indicate the reason the command terminated. The following status codes are specific to DDE.

Table 5-1 DDE Error Messages and Status Codes

L	Indent	Message
E	DDEBADCONN	DDE Bad conversation handle
E	DDEBADDATA	DDE Bad data handle
E	DDEBADDISC	DDE DISCONNECT failed
E	DDEINVDATAL	Invalid data link requested
E	DDEMAXADVISE	Maximum number of advise items reached
E	DDEMAXCONN	Maximum number of connections reached
E	DDENOCONN	DDE CONNECT failed
E	DDENODATA	DDE Data not available from server

In addition to setting the status code, special DDE messages are displayed just before the $STATUS codes on the command line. These messages often provide more information than the $STATUS code messages.

5.4.3 Client Messages

Client messages are all prefixed with “DDE [Client]:”, followed by the message. The messages that can appear when using the DDE client commands are as follows:

Conversation already exists.

The conversation handle passes to the DDE CONNECT command is currently active. Either disconnect the conversation variable or supply a new conversation variable name.

Data not available from server.

The data requested by the client is not available on the server. The variable name may be misspelled or the symbol on the server may be local instead of global.

Disconnected DDE connection.

The DDE DISCONNECT or DDE DISCONNECTALL command removed the conversation(s).

Error creating DDE data handle.

A severe internal error message. Could be caused by low memory conditions.

Error creating DDE string handle.

A severe internal error message which could be caused by low memory conditions.

Error disconnecting from server!

An internal error indicating that DDE DISCONNECT or DDE DISCONNECTALL failed.

Invalid conversation number.

The conversation number no longer exists. This usually occurs because a DDE CONNECT was not previously complete, or the conversation has terminated already.

Invalid data link requested.

The DDE ADVISE command failed because the item could not be found.

No such data link exists.

The DDE UNADVISE command failed because there was not an active Advise Data Link for this item.

Ok establishing DDE connection.

The DDE CONNECT command succeeded. This message is informational only.

The server forced a disconnect.

The server sent a DDE terminate message during a conversation. The conversation number associated with this connection is no longer valid.

Unsuccessful connection.

The DDE CONNECT command failed. The service name or topic name may be incorrect.

5.4.4 Server Messages

Server messages are all prefixed with "DDE [Server]:", followed by the message. The messages that can appear when the emulator is a server are as follows:

Advised client of change.

The value of an ECL symbol that has an Advise Data Link has changed, and the client was notified. This message is informational only.

Could not create data handle.

A severe internal error message. This may be a result of low memory.

The client has disconnected.

The client application sent a disconnect message to the server and therefore terminated the conversation. This message is informational only.

DDE connection confirmed.

When a client application sends a connect message and the server responds that the connection can be made, the client sends this additional message to confirm the conversation This message is informational only.

Requested data sent to client.

The client application sent a DDE request message to the server. The message was processed and the value of the item was sent to the client. This message is informational only.

Received POKE data from client.

The client application sent a DDE poke message to change the value of an ECL variable. The message was processed and the item was updated with the new value. This message is informational only.

For more information about the emulator error handling, refer to the *Error Facility* topic in Chapter 8.

5.5 DDE COMMAND BUILDER

Click on ***Execute - DDE Command Builder***. The DDE Command Builder makes it easier to perform DDE commands because you don't need to learn the format of each command. In addition, each field in the Command Builder contains a list of previously entered DDE parameters. Currently, this list holds up to 10 parameters.

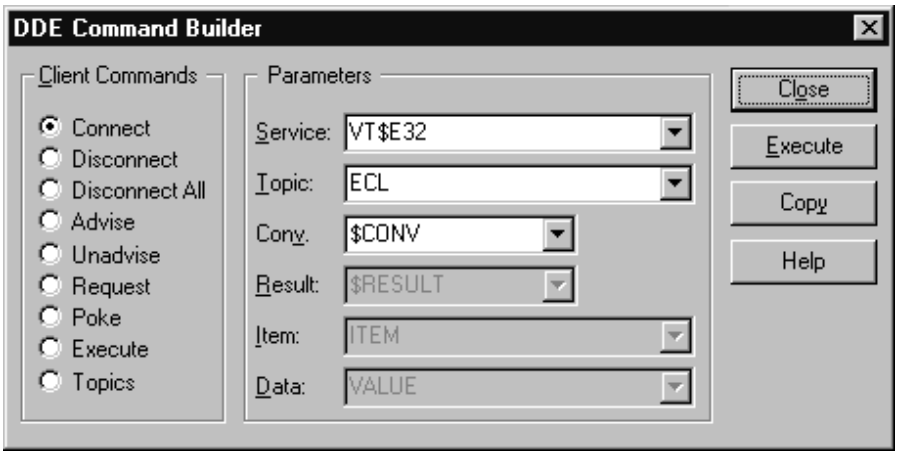


Figure 5-1 DDE Command Builder

The DDE Command Builder is used in the following way:

1) To select the desired DDE command, click on the appropriate button under the **Client Commands** heading. Notice that as different DDE commands are selected, some of the **Parameters** may become enabled or disabled. This indicates the required parameters for the selected command.

2) Once the desired DDE commands are selected, enter or select the data for the Parameters

3) When all parameters are entered, click the Execute button. The emulator creates the DDE command string, copies the string to the command line, then executes the command string. The results display above the command line just as if the commands were entered on the command line.

5.5.1 Copying a DDE Command to the Command Line

To edit the DDE command string before executing, click on the Copy button. The emulator creates the DDE command string with the entered parameters, then copies the string to the DDE command line. You can then edit the string by positioning the typing cursor in the string. When you have finished editing the string, click ***OK*** to execute the command or ***Cancel*** to cancel the command. The Cancel button also closes the dialog box.

5.6 DDE DEMO

The DDE demo demonstrates some of the Dynamic Data Exchange (DDE) capabilities of the emulator. Look at the commands in the command (.ECF) file for examples of how to write your own DDE scripts.

To run the 30 second DDE Demo:

1) Start the emulator, if not already running.
2) Run the Command File DDEDEMO.ECF using the ***File - Run Command File*** dialog box.

The DDE demo displays a screen that indicates the current time and US population. This information is provided by the DDE server.

ASCII CONTROL CODE TABLE

OVERVIEW

The ASCII Control Code Table can be used during Keyboard, Mouse and Toolbar mapping.

ASCII Character	Hex Code	Decimal Code	Keystroke
NUL	00	0	Ctrl @
SOH	01	1	Ctrl A
STX	02	2	Ctrl B
ETX	03	3	Ctrl C
EOT	04	4	Ctrl D
ENQ	05	5	Ctrl E
ACK	06	6	Ctrl F
BEL	07	7	Ctrl G
BS	08	8	Ctrl H
HT	09	9	Ctrl I
LF	0A	10	Ctrl J
VT	0B	11	Ctrl K
FF	0C	12	Ctrl L
CR	0D	13	Ctrl M
SO	0E	14	Ctrl N
SI	0F	15	Ctrl O
DLE	10	16	Ctrl P
DC1	11	17	Ctrl Q
DC2	12	18	Ctrl R
DC3	13	19	Ctrl S
DC4	14	20	Ctrl T
NAK	15	21	Ctrl U
SYN	16	22	Ctrl V
ETB	17	23	Ctrl W
CAN	18	24	Ctrl X
EM	19	25	Ctrl Y
SUB	1A	26	Ctrl Z
ESC	1B	27	Ctrl [
FS	1C	28	Ctrl \
GS	1D	29	Ctrl]
RS	1E	30	Ctrl ^
US	1F	31	Ctrl _

INDEX

!

A

B

C

D

E

Q

R

S

T

U

V

W