MP/AOS

Slate Text Editor

Data General

MP/AOS

SLATE Text Editor

Data General

Notice

Ordering No. 069-400209

Rev. 00, July 1982

Preface

This manual is divided into five chapters. Chapter 1 is a general introduction to SLATE operations. Chapters 2, 3, and 4 document the functions that can be performed in each of the primary operating modes of SLATE: page edit, command, and key definition. Chapter 5 covers more advanced functions that can be used to program iterative operations.

Appendix A describes the FIND utility, which is similar in many respects to the SEARCH and CHANGE commands described in Chapter 3. Appendix B shows the identification symbols for the function keys.

Command-Line Conventions

Throughout this manual, we use the following conventions to illustrate command-line formats:

COMMAND	Upper-case letters indicate a command. Type these commands as they appear on the command line or abbreviate the command name by using the fewest characters that uniquely defines it. Switches also appear in this form if actual switch names are used.
argument	Lower-case italics represent an argument or the general term, *switch*. Replace these terms with the exact name of an argument or switch.
[optional]	Brackets indicate an optional element of the command line. Do not include the brackets in your command line; they only set off the option. (In the case of pseudo-commands and certain template expressions for the SEARCH and CHANGE commands, brackets are part of the command line coding. See Chapters 3 and 5.)
...	Three dots indicate that one or more commands, arguments, or switches can be added to the command line.
↵	A curved down arrow indicates a line terminator.

Related Manuals

The following manuals also belong to the series of books published on the MP/AOS operating system.

MP/AOS Concepts and Facilities (DGC No. 069-400200) provides a concise but thorough introduction to the MP/AOS operating system for users who want to assess the system's advantages.

MP/AOS System Programmer's Reference (DGC No. 093-400051) documents MP/AOS system structure and provides a complete dictionary of system calls and library routines.

MP/AOS Command Line Interpreter (CLI) (DGC No.069-400201) describes the interactive CLI program, the user's primary interface to the MP/AOS system. A command dictionary provides command descriptions, formats, and examples.

Loading MP/AOS (DGC No. 069-400207) describes how to install MP/AOS software on ECLIPSE-line computers and how to load tailored systems.

MP/AOS System Generation and Related Utilities (DGC No. 069-400206) describes the generation of an MP/AOS system tailored to specific applications. It also describes the following utilities, including sample dialogues as appropriate:

- SYSGEN, the interactive system generation utility;
- DINIT, the disk initializer;
- FIXUP, the disk repair utility;
- SPOOLER, which controls line printer operations;
- ELOG (error logger), the utility for interpreting the system log file.

MP/AOS Debugger and Performance Monitoring Utilities (DGC No. 069-400205) describes the following utilities, providing a dictionary of debugger commands and sample dialogues as appropriate:

- FLIT, the process debugger;
- PROFILE, which measures execution-time performance;
- OPM, the process monitor that displays current system resource allocation and status.

MP/AOS Macroassembler, Binder, and Library Utilities (DGC 069-400210) documents the MP/AOS macroassembler and binder as well as the library file editor (LED) and system cross-reference analyzer (SCAN). It includes programming examples and a dictionary of assembler pseudo-ops.

MP/AOS Advanced Program Development Utilities (DGC 069-400208) describes the following utilities:

- Text control system (TCS), a method for managing different versions of a single file;
- BUILD, which creates a new version of a file from existing files, thus minimizing effort and errors in program development;
- FIND, which locates occurrences of strings in text files.

MP/AOS SPEED Text Editor (DGC No. 069-400202) documents the features of SPEED, the MP/AOS character-oriented text editor.

MP/AOS File Utilities (DGC No. 069-400204) describes the following utility programs, providing sample dialogues for each:

- FEDIT, a file editor that permits modification of system files, including program and data files;
- FDISP, which can display the address and data contents of a file or compare two files, displaying the parts that differ;
- SCMP, which can compare two source programs line by line;
- MOVE, which allows the transfer of files among directories;
- AOSMIC, which allows manipulation of MP/AOS and MP/OS disks and files on an AOS system;
- FOXFIRE, which permits the transfer of files among MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.

SP/Pascal Programmer's Reference (DGC No. 069-400203) documents an extended Pascal for system programmers. SP/Pascal has all of the features of MP/Pascal as well as special features targeted for the MP/AOS and AOS operating systems.

Books on three additional programming languages supported by MP/AOS have previously been published as part of the bookset for the MP/OS operating system:

MP/Pascal Programmer's Reference (DGC No. 069-400031) documents for system programmers a Pascal-based language targeted for the MP/OS operating system.

MP/FORTRAN IV Programmer's Reference (DGC No. 069-400033) documents for system programmers a language based on ANSI 1966 standard FORTRAN with extensions.

MP/Basic Programmer's Reference (DGC No. 069-400032) documents for new users a programming language based on ANSI standard Basic with extensions.

MP/OS

For information on Microproducts and a bibliography of documentation on the Microproducts line, see *Introduction to Microproducts,* DGC No. 014-000685.

For information on cross development between MP/OS and MP/AOS, see *MP/OS System Programmer's Reference,* DGC No. 093-400001.

Contents

Tables

Operating SLATE

1

This chapter explains how to enter SLATE from the CLI and how to exit from the program. It describes the switches that can be used in the command line and the information that appears on the header line of the screen.

The chapter also introduces the concepts of buffers and files (including input and output files), the three primary modes in which SLATE operates, and the keyboard.

Finally, it tells how to get additional information about SLATE from SLATE itself and how to interrupt a console operation.

SLATE Operations

SLATE is a screen- and line-oriented text editor. It has many features that promote efficiency in text editing, besides offering multiple text buffers, the ability to program macros, and the ability to use CLI commands without exiting from the editor.

SLATE editing commands insert, delete, and modify individual characters, words, lines, or blocks of text and search for specific words, strings, or patterns. With SLATE, you can also restore deleted lines, reformat text, and convert text to either upper or lower case.

The logical text buffers provide workspaces for creating, modifying and merging text files. File operations are not restricted to the current working directory.

Macros can be written as key definitions that can be saved in files and re-executed automatically. SLATE allows up to 25 key definitions created by the user to be used at any one time.

Command sequences can also be saved in files and re-executed.

Buffers and Files

A logical text buffer is the temporary location in memory for text that is being edited. This text may be in the form of an existing file, a piece of text copied from another text buffer, or characters entered from the keyboard.

The basic unit of text is a character (any single 8-bit byte). This byte, interpreted as an ASCII character, occupies one column position in the logical text buffer.

The current buffer is the one in which you are working. SLATE allows you to work in any of ten different buffers (numbered 0 to 9).

A body of text is stored in a file. When you open an existing file for editing with SLATE, it is read into the current buffer. When you close a file, the text is removed from the buffer containing it and either written back into storage or deleted.

Input and Output Files

You can type text from the keyboard into the current buffer or transfer text from one buffer to another. Or you can open an existing file for input and read it into the current buffer. You may want to save an input file in the form in which it entered the buffer - that is, before you altered (edited) it. SLATE allows you to do so by giving you the option of saving a backup file (your unedited input file). This option depends on how you opened the file.

Your output file consists of edited text that you want to save. When you close that file, the text is written out from the current buffer into the file.

These concepts are explored further below and in Chapter 3. Refer to *MP/AOS Command Line Interpreter (CLI),* DGC No. 069-400201, for a complete description of file structures and directories under MP/AOS.

Executing SLATE

Type the following CLI command to execute SLATE:

XEQ SLATE[/*switch*...] [*filename*] ⤶

If you are using SLATE under the AOS operating system, type

XEQ MSLATE[/*switch*...] [*filename*] ⤶

where:

XEQ (or X)	is the execute command;
/switch	is one or more of the switches defined in Table 1.1;
filename	is the name of the file you wish to edit.

If *filename* is an existing file, which is not in your working directory or on your searchlist, you must type the full pathname, including *filename*. Including *filename* in the command line not only opens the file for editing but automatically provides for an output file.

You can also execute SLATE without specifying a filename. In that case, you have two options:

1. You can type text into the current buffer. To save your text, you must explicitly create an output file using the WRITEFILE command (see Chapter 3).
2. You can open an existing file for editing by using the OPENFILE or READFILE command (see Chapter 3). OPENFILE automatically creates an output file. With READFILE, you must again explicitly create the output file.

Command-Line Switches

You can use one or more switches on the command line when you enter SLATE. Table 1.1 lists these switches and describes their functions.

All switches begin with a slash (/). You can abbreviate a switch name to the fewest characters that uniquely defines it. For example, the switch /MARKSPACES can be abbreviated to /MAR.

Switch	Effect
/BACKUP	When a file is closed, saves as a backup file the input file (if any) that was opened for editing.
/BUFFERS=*n*	Determines the number of extra text buffers available besides Buffer 0. Up to 9 extra buffers are available. (By default, *n* is 5.)
/I=*pathname*	Executes any commands contained in *pathname.* After executing *pathname,* SLATE takes further command input from the keyboard. This is an initialization switch, which makes available for an editing session the operations defined in key definition or command files (see Chapters 4 and 5). If a filename is specified as an argument to the XEQ command, that file is opened *before* the commands in the initialization file are executed.
/LPP=*n*	Determines the number of lines of text to be displayed on the screen at one time. (By default or if *n* exceeds screen size, a full screen of text is displayed.) This switch is useful at slower data rates.
/MARKSPACES	Causes all spaces (not tabs) to be displayed as dimmed periods. (By default, both spaces and tabs are displayed as spaces.) Compare with /REVERSEDIM, below.
/MAXLENGTH=*n*	Determines maximum length of a line (up to 256 characters). (Default line length is 100 characters.) The lower the value of *n*, the more efficiently SLATE operates.
/NOTABS	Causes all tabs to be converted to spaces, whether part of the input file or typed from the keyboard. (By default, tabs are not converted to spaces.)
/NSL	Prevents the CLI from searching in your search list for the filename you want to edit. This switch has no effect once you have entered SLATE.
/REVERSEDIM	Causes normally bright characters to be dim on the screen and normally dim characters to be bright.

Table 1.1 SLATE command-line switches

The Keyboard

Besides the keys you use for ordinary typing, three kinds of keys are important in using SLATE:

1. Control characters.
2. Preprogrammed function keys, including the two keys that allow you to enter command mode and key definition mode.
3. Undefined function keys.

Keyboards vary somewhat, but control characters are normally formed by pressing the CTRL key on the left of the keyboard in conjunction with another key.

Function keys are the unlabelled row of keys on top of many Data General DASHER-type keyboards. Each function key can be used for up to four different functions by using it alone as well as in combination with the SHIFT and/or CTRL keys. Several of the function keys have been preprogrammed. Others have been left undefined, so individual users can define their own special functions or combine various preprogrammed functions. Chapter 4 explains how to define keys to perform the special functions you require.

SLATE supports the Data General Corporation DASHER line of CRT terminals from the DASHER D2 (model 6053), the DASHER D200 (models 6108 and 6109), and the DASHER D400 and D450 (models 6130 and 6134). The DASHER D2 keyboard configuration differs somewhat from the other models listed above. It has only 11 function keys, while all the other models have 15. Function key templates for both types of keyboards are available (part no. 069-400047 for the DASHER D2 and part no. 069-400046 for the other models).

Operating Modes

You can operate SLATE in one of three primary modes: page edit, command, and key definition. These are introduced briefly below and described in detail in Chapters 2, 3, and 4, respectively.

Page Edit Mode

When you enter SLATE, you are in page edit mode, the "home mode" in which you enter text into files and edit those files. You enter the command and key definition modes from page edit mode and normally you return to it after performing an operation in one of the other two modes. In general, the functions you perform in page edit mode are

moving the cursor;
inserting and deleting text and whitespace;
moving the page (screen);
setting margins and tab stops.

These functions are performed directly at a keystroke or a combination of keystrokes. The keys you use are *control characters* or *function keys.* You do not type in commands in page edit mode.

Command Mode

In command mode, you use command keywords instead of keystrokes to specify the function you want to perform.

You enter command mode from page edit mode by pressing the Command function key (key F1). You can then type in a command keyword next to the command mode prompt (>) that appears on the bottom of the screen. The command is executed when you type New Line.

Normally, when the command is executed, you automatically return to page edit mode. Sometimes, however, you are given a second command mode prompt. You can then execute another command or return to page edit mode by typing New Line.

Many of the commands executed in command mode perform functions similar to those you can perform in page edit mode. Other functions are specific to command mode. In general, these are

opening, creating, and closing files;
manipulating files and buffers;
manipulating text strings;
programming command sequences and iterative operations.

Key Definition Mode

In key definition mode, you can use predefined keys, command keywords, and text to define keys. You enter key definition mode from page edit mode by pressing the Define key function key (CTRL-F1). You can then define or redefine function keys to perform specific functions.

Header Line

A header line extending across the top of the screen gives information regarding page edit mode and your editing activities in this mode. The header line contains the following:

- On the left, a message field which may indicate the revision (rev) number of the SLATE program you are using, or the operation in progress (delete, copy, transfer, etc.), or an error message or warning.
- Breaks in the header line itself, which indicate tab stop locations.
- On the right, a character or characters indicating when a screen edit mode has been turned on. These characters are as follows:

C	CTRL Character Shift flag
H	Hi Bit Shift flag
T1 or T2	Mark Text flag
B1 or B2	Mark Block flag
CTRL-P	CTRL-P flag
CTRL-J	Line insert mode
CTRL-E	Character insert mode

The meaning of the functions highlighted by these indicators will become clear as you read through this manual.

HELP Command

To display a list of SLATE commands and topics relating to the SLATE text editor, enter command mode and type next to the command prompt

```
> HELP ↵
```

To display information on a specific command, type

```
> HELP[/V] command ↵
```

where *command* is a command mnemonic and /V is a switch generating more complete information than is displayed when this switch is not used.

To display information on a specific topic, type

```
> HELP *topic ↵
```

where **topic* is one of the topics displayed when you type HELP without arguments.

You can control the scrolling of the information on the screen by typing CTRL-S to freeze the screen and CTRL-Q to resume scrolling. (If you are using SLATE under the AOS operating system, press the BREAK key before typing CTRL-S or CTRL-Q.)

Console Interrupts

It is sometimes useful to be able to interrupt runaway searches or other operations that were performed by mistake or that take a long time to complete. SLATE has a console interrupt facility, executed by typing the control character sequence, CTRL-C, CTRL-A. (Under the AOS operating system, you must press the BREAK key first.)

When a console interrupt is executed, SLATE stops what it is doing, resets itself, and waits in the current editing mode for further input.

Exiting from SLATE

To exit from an editing session, simply press the Command function key and type out BYE ↲ next to the SLATE prompt (>). This saves your edited text in an output file (if an output file has been opened) and returns you to the CLI.

BYE accepts two switches, /BACKUP and /IGNORE.

If you type

```
    > BYE/BACKUP ↲
```

your original input file is saved as a backup file, providing you opened the file with some form of the OPENFILE command described in Chapter 3. (When you execute SLATE by specifying an existing file with the *filename* argument, an OPENFILE command is automatically executed.)

As we have seen, you can also use the /BACKUP switch when you execute SLATE. In this case, a backup file of your input file (if any) is automatically saved, whether or not you use the /BACKUP switch on exiting from SLATE.

The /IGNORE switch requires some explanation.

SLATE allows you to work in any of ten different buffers. When you execute SLATE, you automatically begin to work in Buffer 0. From Buffer 0, you can transfer or copy a block of text to any other buffer and edit it there. Or you can create new files or open existing files in any buffer.

SLATE prevents you from inadvertently losing any files in buffers other than the current one (the buffer you are working in). If you have any open files in noncurrent buffers, you can only exit from the program by typing

```
    > BYE/IGNORE ↲
```

SLATE then ignores any files in noncurrent buffers and writes the text in the current buffer to the output file (if an output file exists). If you simply type BYE ⤶, an error message is displayed, as follows:

```
Error: Non-current output file(s) open, see /IGNORE
```

You can then type BYE/IGNORE ⤶. Or, if you want to save an open file, you can either close it (with the CLOSEFILE command) or consolidate it in the current buffer and then type BYE ⤶.

See Chapter 3 for a description of the CLOSEFILE command.

NOTE: *You do not have to exit from SLATE in order to execute CLI commands. See the XCLI command, described in Chapter 3.*

Page Edit Mode

2

As soon as you invoke SLATE from the Command Line Interpreter (CLI), you are in page edit mode. This is "home mode," in which you enter your text and perform many editorial functions. In general, the functions you can perform in page edit mode are:

- moving the cursor;
- moving to a different page (screen);
- inserting and deleting text and white space;
- setting margins and tab stops.

You also enter the other two primary modes (command and key definition) from page edit mode, and you can perform some miscellaneous functions. All these functions are performed via control characters or function keys, that is, with a keystroke or combination of keystrokes.

This chapter describes the functions you can perform in page edit mode.

Text Concepts

To be able to use most effectively the editing functions described in this and subsequent chapters, you should be familiar with a few of the concepts involved in handling text with a text editor.

Line Elements

As Chapter 1 explains, the basic unit of text is the *character* (any single 8-bit byte), which occupies one column position in the logical text buffer and is interpreted as an ASCII character.

A *character string* is a sequence of characters. The sequence can include any ASCII character, including control characters, non-printing characters, and characters with high bits set.

Whitespace is any character string consisting only of spaces or tabs. In some contexts, the definition extends to include New-Line characters.

A *line* is a string of characters ending with a line delimiter (usually New Line) or that is MAXLENGTH characters long (see Table 1.1) or that ends at the end of the logical text buffer. Maximum line length is 256 characters.

A *line terminator* is a New Line or Form Feed. Unless the /REVERSEDIM switch has been used on the SLATE command line (see Table 1.1), a New Line is displayed as a dim J (CTRL-J) if the line has trailing whitespace, and a Form Feed is displayed as a dim L (CTRL-L). If a line has no terminator - i.e., if it has been truncated by the /MAXLENGTH switch - it is marked with a dim |. No line terminator is displayed until the cursor moves to another line.

Using Tabs

SLATE offers two ways to handle tabs. One is to convert all tabs to spaces by using the /NOTABS switch on entering SLATE. This method has the drawback of not preserving tabs in the output file.

When a file is opened without the /NOTABS switch, the tabs in the text are converted to the appropriate number of 1-space "tiny tabs." For example, a standard 8-space tab is converted into eight 1-space tiny tabs.

When a tab is entered from the keyboard in page edit mode (or in INSERT strings in command mode), enough 1-space tiny tabs are inserted to reach the next tab stop. However, the entire group of tab characters moves left or right when preceding characters on the line are deleted or inserted. In other words, tab stops are not observed when a line is altered.

A single tiny tab can be entered with the sequence CTRL-P Tab (see Table 2.4).

For most practical purposes, you can think of tiny tabs as spaces. That is, you type over, delete, and move the cursor across them individually.

When the file is closed, all "tiny tabs" are grouped and converted back into single 8-space standard tabs wherever possible and to spaces when this is not possible.

Screen Scrolling

Normally, the right margin is located near the right edge of the screen, but it can be set beyond the edge. Also, the right margin of a line may be released and will remain released while the cursor stays on that line. The screen scrolls right when the cursor moves beyond the right edge of the screen. It scrolls left again when the cursor is moved beyond the left edge of the screen.

The screen scrolls up several lines when the cursor reaches the bottom line of the screen. If screen is not displaying the top line of the buffer, it scrolls down when the cursor is at top line of screen and you move cursor up.

Using Page Edit Mode

With the foregoing as background, we can now get into the essence of editing text in page edit mode. We begin with an explanation of functions performed with control characters. This is followed with a discussion of functions performed with function keys.

NOTE: *Many of the editing functions described in this chapter can also be performed on the command lines in command mode and key definition mode (see Chapters 3 and 4). However, keystroke functions that move the cursor outside the current line can only be used in page edit mode. Also, the delimiting operations described in this chapter (see Tables 2.6 and 2.7) are performed by different means in command mode. Other changes in function from mode to mode are noted in the tables that follow. Some practice with SLATE will clarify this for you.*

Control Characters

A control character is formed by pressing the control key (CTRL on left of keyboard) and another key. Tables 2.1 and 2.2 list the control characters you can use to move the cursor and insert and delete characters and lines.

Depending on the keyboard you are using, several of the operations described in these tables can also be performed using other keys on the keyboard. For example, CTRL-J is identical to New Line. Duplication of key functions is noted in the tables.

Control Character	Function
CTRL-A	Moves cursor to end of current line of text, moving from left or right. If cursor is already at end of line, it moves to end of next line.
	In command mode, recovers characters of previous command.
CTRL-B	Moves cursor to space following previous word. If there is no previous word, moves cursor to first character position in buffer.
CTRL-F	Moves cursor to beginning of next word. If there is no next word, cursor stops after last character in buffer.
	In command mode, recovers characters of previous command.
CTRL-H	Identical to HOME key. Moves cursor to home position on screen (left margin, top line). If cursor is at home position, moves it to left margin, bottom line.
CTRL-I	Identical to Tab key. Moves cursor to next Tab stop. Inserts tiny tabs when you tab beyond end of line and then type a character.
	When CTRL-E is ON (see Table 2.2), inserts tiny tabs up to next tab stop as cursor moves to that stop.
	In Key Definition mode, enters a single tiny tab.
CTRL-J	Identical to New Line. Moves cursor to left margin, down one line. At end of logical text buffer, no New Lines are inserted unless another character is typed.
	If line insert mode is on (see Table 2.5), splits line at cursor, creating an extra line. If there are spaces or tabs at the end of a line, a CTRL-J appears on screen to show position of New Line.
CTRL-M	Identical to Carriage Return. Moves cursor to left side, *current* line.
CTRL-W	Identical to cursor-up key (↑). Moves cursor up one line to same column position. If this brings cursor beyond end of line, the line is not extended with tiny tabs unless a character is typed.
	If cursor is at top of screen, screen scrolls downward, but not past start of buffer.
CTRL-X	Identical to cursor-right key (⟶). Moves cursor right one character. If this brings cursor beyond end of line, the line is not extended with a tiny tab unless a character is typed. If cursor is at right edge of screen, screen scrolls right.
	In command mode, recovers characters of previous command.
CTRL-Y	Identical to cursor-left key (⟵). Moves cursor left one character. If cursor is at left of screen (but not at column 1), screen scrolls left. If cursor is at column 1, CTRL-Y has no effect.
CTRL-Z	Identical to cursor-down key (↓). Moves cursor down one line to same column position. If this brings cursor beyond end of line, the line is not extended with tiny tabs unless a character is typed.

Table 2.1 Control characters: moving cursor

Control Character	Function
CTRL-E	Turns character insert mode on or off. Initially off. OFF - Any character typed replaces character at cursor location ON - A CTRL-E appears on hearder line of screen. Any character typed moves all characters at and to right of cursor one space right. When ON, and CTRL-I (CTRL-Tab) is typed, tiny tabs are inserted up to next tab stop as cursor (and characters to right of cursor) move to (and beyond) that stop.
CTRL-K	Identical to ERASE EOL key. Erases characters from cursor to end of line, except for line delimiter.
CTRL-U	Erases entire current line, regardless of cursor position, without deleting the line itself (i.e., leaving whitespace), moving cursor to left margin. Erased text may be restored with Undelete Line (see Table 2.5), except in key definition mode.
DEL	Deletes character to left of cursor and closes up line.

Table 2.2 Control characters: inserting/deleting characters/lines

As you can see already, there are several ways you can approach the problem of inserting or deleting text. Besides using the control characters listed in Table 2.2, you can use the Space Bar to replace text at and ahead of the cursor with whitespace. You can also use function keys to perform related functions (see Table 2.5). With practice, you will find the easiest way for you in different circumstances.

Besides performing the functions listed in Tables 2.1 and 2.2, you can use control characters to insert form feeds and perform a few miscellaneous functions (see Table 2.3).

Sometimes two control characters are used in sequence to perform a specific function. You have already encountered one of these control character sequences: the CTRL-C, CTRL-A sequence that performs a console interrupt. Several control character sequences are listed in Table 2.4.

Control Character	Function
CTRL-D	No function in page edit mode or key definition mode. In command mode, terminates a line.
CTRL-L	Replaces original line terminator with form feed to indicate where page is to end when printed. A CTRL-L appears on screen where form feed is positioned. When line insert mode is ON (see Table 2.5), a form feed is inserted at cursor position or before original line terminator (whichever comes first), splitting original line. To replace a form feed with a New Line, use CTRL-P, CTRL-J with line insert mode turned off (see Table 2.4).
CTRL-P	Accepts a character or function key/character combination without interpretation. (See Chapters 4 and 5 for significance of this function.) For effect with CTRL-I, CTRL-J, CTRL-L, see Table 2.4.

Table 2.3 Control characters: form feed and miscellaneous

Sequence	Function
CTRL-P, CTRL-I	Enters a single tiny tab at cursor position. Equivalent to CTRL-P, Tab.
CTRL-P, CTRL-J	Places New Line in text buffer, replacing original line terminator.
	When line insert mode is ON, line is split at cursor position, forming two lines. The first line has the new terminator and the second has the original terminator.
CTRL-P, CTRL-L	Equivalent to CTRL-L.
CTRL-Char Shift, |	Removes line terminator from a line, so that the line continues on the next "line."
	When line insert mode is ON, line is split at the cursor position, forming two lines. The first line has no terminator and the second (a continuation of the first) has the original terminator.

Table 2.4 Control character sequences

Function Keys

The function keys are the row of unlabelled keys located on the top of the keyboard. They are divided into two main groups: undefined keys and preprogrammed keys.

The function keys (named F1, F2, etc. up to F11 on some keyboards and up to F15 on others) can be made to perform four different functions each. This is accomplished by using various combinations of the SHIFT and CTRL keys to the left of the keyboard. They can be combined with the function keys in the following ways:

1. Use a function key by itself.
2. Use the SHIFT key and a function key at the same time.
3. Use the CTRL key and a function key at the same time.
4. Use the CTRL key, the SHIFT key, and a function key all at the same time.

Combining these keys in this way allows you to perform up to 60 different functions (providing your keyboard has 15 function keys). In this chapter, we describe the 25 preprogrammed functions you can perform with the function keys.

Inserting, Opening, Deleting/ Undeleting Lines

Five of the preprogrammed function keys insert, open, delete and undelete lines and close whitespace. These are listed in Table 2.5. Compare the functions listed in this table with those listed in Table 2.2, which shows related functions performed by control characters.

Function Key	Function
Line Insert (CTRL-F2)	Turns line insert mode on or off. Initially off. When ON, press New Line to break line at cursor, moving remainder of line to left margin, next line. Pressing New Line again opens another line. A CTRL-J appears on header line of screen.
Open Line (F2)	Opens a line by breaking line at cursor position. Remainder of line moves down one line to *same* column position.
Close Whitespace (F3)	Deletes whitespace between cursor and next character to right.
Delete Line (CTRL-F6)	Deletes entire current line. Cursor moves to left margin up one line, closing up space below.
Undelete Line (CTRL-F7)	Places at cursor position most recently deleted line (whether deleted in page edit or command mode and whether deleted with Delete Line or CTRL-U). In page edit mode, current line moves downward. Otherwise, current line is overlayed by undeleted line.

Table 2.5 Function keys: inserting, opening, deleting/undeleting lines

Operations on Delimited Text

The Mark Text and Mark Block function keys delimit (mark) blocks and strings of text for various operations.

A text block is an *m* x *n* rectangle of text. If a rectangle were drawn anywhere on a screen full of text, the characters enclosed within the rectangle would form a block of text. Blank spaces are retained when the block is moved. When you move a block of text to another buffer, and then enter that buffer (making it the current buffer), the block attribute of the text is lost.

A text string is a section of text viewed consecutively, line by line, from one arbitrary point to another. In delimiting operations, this term is used in opposition to text block.

Using either the Mark Text or Mark Block function key in combination with an operation code you type from your keyboard, you can

- erase or delete portions of text;
- create whitespace of a certain size in a body of text;
- convert portions of text to upper- or lower-case letters;
- reformat text by squashing out whitespace and justifying it with the right margin;
- copy or transfer parts of your text to another buffer.

To mark text for one of these operations, you must depress either the Mark Text or Mark Block function key twice. Press it once to mark the cursor position for one end of the text block or string and once to mark the other end.

Text strings (delimited with the Mark Text key) start at the position of the first mark and go up to but do not include the second mark. Text blocks (delimited with Mark Block key) are marked at and include any two diagonally opposed corner positions.

When you have marked the beginning and end of the text block or string, you must type an operation code or abort the operation with New Line.

When you have copied or transferred text to another buffer with Mark Text or Mark Block, you can insert or overlay this text into the current buffer by using the Mark Position function key.

Table 2.6 lists the three function keys you can use to mark text. Table 2.7 lists the codes for the operations you can perform on delimited text.

Function Key	Function
Mark Text (F4)	Delimits text for *string* insertion or other operation. Depress key twice, for beginning and end of string. T1 appears on header line of screen for beginning of string and T2 for end of string. Then, type in a code for the operation you want to perform on marked text (see Table 2.7), followed by New Line. Press New Line to abort operation before entering a code.
Mark Block (F5)	Delimits text for *block* insertion or other operation. Depress key twice, for any two diagonal corners of rectangular block of text. B1 appears on header line of screen for first corner and B2 for second. Then, type in a code for the operation you want to perform on marked block (see Table 2.7), followed by New Line. Press New Line to abort operation before entering code. A block of text transferred into another buffer loses its block attribute as soon as the buffer containing it becomes the current buffer.
Mark Position (SHIFT-F4)	Marks position for insert or overlay (text block or string) from another buffer. Depress key; then type in one of the following operation codes, followed by New Line. I*n* — Insert text from buffer *n* O*n* — Overlay text from buffer *n* If text is in block form, upper left corner appears at cursor position. Press New Line to abort operation before entering code.

Table 2.6 Function keys: marking text and inserting/overlaying marked text

Operation Code	Operation Performed
D	Deletes delimited text. If text string is deleted, text is closed up. If text block is deleted, text to right overlays deleted block.
E	Erases delimited text, leaving whitespace.
W	Opens whitespace the size of delimited text. If text string is marked, delimited text follows whitespace. If text block is marked, delimited text moves to right of block of whitespace.
L	Converts letters within delimited text to lower case.
U	Converts letters within delimited text to upper case.
S	Squashes extra whitespace out of delimited text strings (not blocks). Changes multiple spaces and tabs to single spaces (double spaces at end of sentence). Squashed text is confined to space between left and right margins. Paragraph indentations are maintained, as is any indentation for second line of a paragraph. Subsequent lines line up with second line of paragraph.
J	Justifies delimited text. Same as S (squash) except extra spaces are placed between words where necessary so that affected lines are justified with right margin. A line will not be justified if more than three extra spaces per word are required.
C*n*	Copies delimited text to buffer *n*, destroying any previous contents. If buffer *n* has open files, an error message is displayed, as follows: `Error: Buffer has open file(s)`
T*n*	Transfers delimited text to buffer *n*. Same as C*n* (copy) operation, except transferred text is deleted from current buffer.
New Line	Cancels operation on delimited text

Table 2.7 Operation codes for delimited text

Other Preprogrammed Functions

Seven of the preprogrammed function keys are for setting and releasing margins and tabs (see Table 2 8). Five function keys move pages on and off the screen (see Table 2.9). The five remaining function keys enable you to enter command mode and key definition mode, back tab, and perform two other special functions. These are listed in Table 2.10.

Function Key	Function
Left Margin (CTRL-F4)	Sets left margin at current column.
Right Margin (CTRL-F5)	Sets right margin at current column.
Margin Release (SHIFT-F5)	Releases right margin, allowing text to be entered beyond it.
Set Tab Stop (F6)	Sets tab stop at current column.
Clear Tab Stop (F7)	Clears tab stop at current column.
Default Tabs (SHIFT-F6)	Clears all tab stops and then sets one tab stop every eighth column, beginning with first column.
Clear All Stops (SHIFT-F7)	Clears all tab stops.

Table 2.8 Function keys: setting margins and tabs

Function Key	Function
Page Up (F9)	Moves one page up in buffer and displays it on screen.
Page Down (F10)	Moves one page down in buffer and displays it on screen.
Page Left (SHIFT-F9)	Moves one page to left in buffer and displays it on screen.
Page Right (SHIFT-F10)	Moves one page to right in buffer and displays it on screen.
Redraw (SHIFT-F1)	Redraws the screen, centering current line.

Table 2.9 Function keys: moving pages on and off screen

Function Key	Function
Command (F1)	Puts SLATE into command mode. Command prompt (>) appears on bottom of screen.
Define Key (CTRL-F1)	Puts SLATE into key definition mode. The word "Define:" appears on bottom of screen.
Back Tab (F8)	Moves cursor to previous tab stop.
CTRL Char Shift (CTRL-F8)	Converts the next character to a control character.
High Bit Shift (SHIFT-F8)	Sets the high bit on the next input character that is not part of a function key.

Table 2.10 Other function key operation

Command Mode

3

Command mode gives you another means of performing several of the editing functions you can perform in page edit mode. It also allows you to perform several important additional functions, as follows:

- opening, creating, and closing files;
- moving text between buffers;
- searching for and replacing text strings;
- executing CLI commands without exiting from SLATE;
- programming iterative, conditional, and counting operations.

This chapter describes all the functions you can perform in command mode, except for the programming or automated editing operations. These are described in Chapter 5.

The commands in command mode are performed via command keywords typed at your terminal. Function keys and control characters are not used to enter commands, but they may be used to edit the command line itself.

Command-Mode Functions

The commands described in this chapter perform the following functions:

- manipulate files and buffers;
- control cursor movement and set and clear tab stops and margins;
- delimit text for various operations;
- insert and format text;
- search for and replace text strings;
- execute CLI commands without exiting from SLATE.

As explained in Chapter 2, page edit mode functions are performed at the stroke of a key or combination of keys. In contrast, command mode functions are performed by typing in command keywords at the keyboard.

The Command Line

To enter command mode, press the Command function key (F1). The command prompt (>) appears at the bottom of the screen. The bottom part of the screen is cleared, if necessary, to make room for the prompt. (When the command is executed and you return to page edit mode, your text reappears, overwriting the command.) Type in your command next to the prompt.

Command-Line Syntax

A command line is formatted as follows:

> COMMAND[/*switch*...] [*arg*[,*arg*]...][;COMMAND...] ↓

Elements of the command line can be entered in upper-case or lower-case letters. Commands and switches can be abbreviated to the fewest number of characters that retains uniqueness, often the first letter of the word. For example, DELETE/BLOCK can be written D/B.

Note that commas separate arguments and a semicolon separates commands. Command lines may not exceed one screen width or one line (whichever comes first).

Pressing New Line or CTRL-L (Form Feed) or CTRL-D executes the command. (Note that, in this manual, a curved down-arrow (↓) stands for any of these line terminators.) When the command is executed, you normally return to page edit mode with the cursor in the same position as previously.

In some cases (for example, commands asking for file or buffer status), you are given a second command prompt after the command is executed. This prevents the screen from being redrawn immediately, allowing you time to read the information returned by the program. It also allows you to type in further commands. (Status commands are usually a prelude to some other operation.) Pressing New Line again returns you to page edit mode.

You also get a second command prompt when you get an error message after attempting to execute a command.

Switches and Arguments

Several switches can be used in command mode. Different commands take different switches. The tables in this chapter indicate the switches that can be used with each command. The switches used for search operations are described in Table 3.5.

Most arguments do one or a combination of the following:

- identify a filename (or pathname if *filename* is not in your current directory or on your searchlist);
- define the second mark for a text operation (the first mark is defined by the current position);
- identify the buffer to receive text or from which text is to be retrieved.

Control Characters and Function Keys in Command Mode

As explained in Chapter 2, many of the editing functions you can perform in page edit mode can also be performed in command mode on the command line itself.

The easiest way to abort a command before executing it is to erase the command line, using CTRL-U or the Delete Line function key. (Undelete Line recovers the previous command line that was erased.)

You can also interrupt a command in the process of execution with a console interrupt (CTRL-C, CTRL-A or, if operating under AOS, BREAK CTRL-C, CTRL-A). SLATE then stops what it is doing, resets itself, and waits in the current editing mode for further input from you. This is useful mainly for lengthy searches or other commands that take some time to execute.

File and Buffer Commands

As a set, the commands that manipulate files and buffers allow you to

- create new files or access existing files and open them for reading or editing;
- distribute parts of the text among ten different buffers to allow you to edit distinct sections more efficiently;
- close the file and return it to its position in the directory.

Individually, the file commands create, open, close, save and abort files and display the file status. The buffer commands select the buffer in which you want to work (which then becomes the current buffer), deactivates noncurrent buffers, and display the status of specified buffers.

The only arguments to these commands are *filename* (or *pathname,*) and *bu_exp* (for a buffer number from 0 to 9 or an expression that resolves to a buffer number).

NOTE: *An expression is usually an integer, but it may also consist of one or more operands and operators, which resolve to an integer. Chapter 5 lists legal operators and operands.*

When you enter a buffer for the first time, the cursor is located at the home position. When you leave a buffer and then re-enter it, the cursor remains in the position it was when you left the buffer.

Input and Output Files

Text can be entered into the current buffer from the keyboard. You can also copy text from one buffer to another. Or you can open an existing file for input (either on entering SLATE or by executing OPENFILE or READFILE). The contents of this input file are read into the current buffer, where you can begin to edit.

Your output file consists of edited text you want to save. When you close this file, text from the current buffer is written out to it (unless you deleted your text with ABORTFILE).

When you open an existing file, using some form of the OPENFILE command, an output file, temporarily called *filename.TM,* will be automatically created. When you close this file, it is given the name of the original input file.

If you used the backup switch (/BACKUP) on entering SLATE, or if you append this switch to a command that closes the file, the input file is saved as *filename.BU.*

When you enter text directly into a buffer, you must create an output file if you want to save your text. In this case, you must use two commands: WRITEFILE, which creates the file, and CLOSEFILE or BYE, which closes it.

NOTE: *When an input file is opened with some form of the OPENFILE command, the output file will have the same file attributes as the input file. When there is no input file, the output file receives the default file attributes. An input file may not be read protected. It can be opened only with READFILE if it is a permanent file or is write protected.*

Table 3.1 lists the commands that manipulate files and buffers. Note that all the commands that manipulate files operate on the current buffer.

Command	Function
OPENFILE *filename*	Opens *filename* for input, copies file into current buffer, overwriting any previous contents, and creates output file. Disallows further file opening or closing commands (except SAVEFILE) until CLOSEFILE or ABORTFILE is executed.
READFILE *filename*	Opens *filename* for input and copies file into current buffer, overwriting any previous contents. Does not create output file, which can be created with WRITEFILE.
WRITEFILE*[/sw] filename*	Creates *filename* for output. Does not affect text in buffer. *Filename* must not be an existing file, unless /APPEND or /DELETE switch is used. If /APPEND is used, new text is appended to existing file. If /DELETE is used, the existing file is deleted and new file replaces it.
CLOSEFILE[/BACKUP]	Writes any text in current buffer to output file, leaving buffer empty. Closes input file, if any, and output file. If used after OPENFILE, input file can be saved by using /BACKUP switch. If there is only an input file, then closes that file and erases buffer, ignoring /BACKUP, if specified. (In this case, CLOSEFILE is equivalent to ABORTFILE.)
SAVEFILE[/BACKUP]	Executes CLOSEFILE command and then opens same file again (with OPENFILE command). Cursor remains in same line and column position. In effect, this command creates an intermediary back-up file without disrupting editing process. Use /BACKUP switch to save original input file as additional backup.
ABORTFILE	Erases text buffer and does not write out text to output file. Closes input file. Deletes output file, if any.
FILESTATUS	Displays pathname of input file for current buffer if files were opened with OPENFILE. Otherwise, displays input and output filenames, if any.
BUFFER *[bu_exp]*	Without argument, displays name of current buffer (0 to 9). Can also be used as a variable (see Chapter 5). With argument, changes current buffer to buffer specified by *bu_exp*.
BKILL *bu_exp...*	Deactivates buffer(s) specified by destroying contents. *Bu_exp* cannot specify current buffer.
BSTATUS *[bu_exp...]*	Without arguments, displays status of active buffers. Otherwise, displays status of buffers specified by arguments.

Table 3.1 Manipulating files and buffers

Opening and Closing Files

As Table 3.1 makes clear, there are several ways to open a file for input. You can also type directly into the buffer and create an output file. Depending on whether or not you opened a file for input and on *how* you opened it, your output file is either created automatically or with the WRITEFILE command. The following list presents several options.

1. Open an existing file by entering SLATE with *filename* argument, thus automatically executing OPENFILE and creating an output file.
2. Enter SLATE without argument and then open an existing file with OPENFILE, automatically creating an output file.
3. Enter SLATE with *filename* argument, when *filename* does not yet exist. Type into the buffer and create output file with WRITEFILE.
4. Enter SLATE without argument and then open an existing file with READFILE. Create output file with WRITEFILE.
5. Enter SLATE without argument and type into the buffer. Create output file with WRITEFILE.

You also have several ways of closing an output file. (Closing the file automatically saves it, unless you delete it with ABORTFILE.) The closing commands can be used regardless of how the output file was created. In all cases, the input file is also closed (and saved, if you used some form of the OPENFILE command to open the file and the /BACKUP switch when you entered SLATE or executed the closing command).

1. Execute BYE and return to the CLI.
2. Execute CLOSEFILE and remain in current buffer, which is now empty. You can then create or open another output file in the buffer.
3. Execute SAVEFILE, leaving the same file in the current buffer with the cursor in the same position as previously. In effect, this gives you an intermediary backup file without disturbing the editing process.
4. Execute ABORTFILE, which closes the input file, if any, erases text in buffer without writing it out to the output file, and deletes the output file. You can then open the same input file again and try once more to edit it.

File Status

Because of all these options, the FILESTATUS command is useful to remind you, if necessary, where you are. If you opened an existing file on entering SLATE or by executing OPENFILE, the FILESTATUS command gives you the pathname of the file. Otherwise, it displays the names, if any, of input and output files. For example:

```
> OPENFILE FOO; FILESTATUS ⤶
I/O: @DPD0:FOO
> ABORTFILE; FILESTATUS ⤶
Input: None
Output: None
> READFILE FOO; FILESTATUS ⤶
Input: FOO
Output: None
```

Buffer Status

Since you can work in any of ten different buffers, you also need some way of moving around among the buffers without losing track of where you are. The BUFFER command (without an argument) tells you which buffer you are in at any time. The same command with an argument allows you to move to another buffer. BKILL allows you to deactivate any buffer except the current one. Finally, the BSTATUS command (without an argument) displays the status of all active buffers. With an argument it displays the status of the buffers you specify. For example:

```
> BSTATUS ⤸
 => Buffer 0   Txt     8911   File(s) open
      Buffer 3   Blk     6 x 20
> BSTATUS 9 ⤸
      Buffer 9   Inactive
```

Note the following:

1. An arrow => points to the current buffer.
2. If the buffer contains text, the number of lines of text in the buffer is displayed (if this information is known).
3. If the buffer contains a block, the dimensions of the block in lines-by-characters is displayed.
4. If a buffer contains open files, that fact is indicated.

Positioning Cursor, Tabs and Margins

The cursor is positioned by line and column in the text buffer. (A column is equivalent to the width of a character position.) The commands listed in Table 3.2 display the current line and column positions, move to a line or column (or intersection of a line and column), identify or set margins, and set or clear tab stops.

There are two arguments to commands in this group. *Li_exp* represents a line number from 1 to 2^{31} or an expression that resolves to a line number. *Col_exp* represents a column number from 1 to 256 or an expression that resolves to a column number.

For all the commands listed in Table 3.2 (except MOVE), *li_exp* and *col_exp* represent absolute numbers, counting forward from line 1 and column 1. In the case of the MOVE command, *li_exp* and *col_exp* represent numbers relative to the current position. For example:

```
> MOVE 4,-3 ⤸
```

moves the cursor 4 lines down and 3 columns backward.

Tab stops in the buffer are indicated by breaks in the header line. Initially, the first tab stop is located at column 1 and from that point there is a tab stop every 8 columns.

Initially, the left margin is set at column 1 and the right margin is set at column 79. The left margin can be moved to any new column as long as it isn't to the right of the right margin. Similarly, the right margin can be moved to any new column as long as it isn't to the left of the left margin.

Normally, text inserted or reformatted in the buffer conforms to the tab stops and margin settings currently in effect.

Command	Function
LINE *[li_exp]*	Without argument, displays current line number. Otherwise, moves cursor to line number specified by *li_exp,* without changing columns. Can also be used as a variable (see Chapter 5).
COLUMN *[col_exp]*	Without argument, displays current column number. Otherwise, moves cursor to column number specified by *col_exp,* without changing lines. Can also be used as a variable (see Chapter 5).
MOVE *[li_exp[,col_exp]]*	Moves cursor forward or backward by number of lines (and columns) specified by *li_exp* and *col_exp.* A positive line expression moves cursor forward; a negative one moves it backwards. A positive column expression moves cursor right; a negative, left. If command is issued without column argument, cursor is moved to column 1 on line indicated in *li_exp.* If command is issued without either argument, has same effect as MOVE 1. Note that MOVE *li_exp,* 0 ↵ is different from MOVE *li_exp* ↵.
END	Moves current position to end of text buffer.
TABSET *col_exp...*	Sets a tab stop at column(s) specified by *col_exp.*
TABEVERY *col_exp*	Starting at column 1, sets a tab stop at column intervals specified by *col_exp.*
TABCLEAR *[col_exp...]*	Clears the tab stops at columns specified by *col_exp.* If columns not specified, clears all tab stops.
LMARGIN *[col_exp]*	Without argument, displays column number of current left margin. Otherwise, sets left margin at column specified by *col_exp.* Can also be used as a variable (see Chapter 5).
RMARGIN *[col_exp]*	Without argument, displays column number of current right margin. Otherwise, sets right margin at column specified by *col_exp.* Can also be used as a variable (see Chapter 5).

Table 3.2 Positioning cursor, tab stops and margins

Operations on Delimited Text

Chapter 2 describes delimiting operations performed by using the Mark Text and Mark Block function keys in combination with operation codes typed into the terminal. Similar operations can be performed in command mode by typing in the appropriate commands. The current position locates the first mark, the argument(s) to the command locate the second mark, and the command operates on the delimited text between.

All of the commands in Table 3.3 operate on text that has been delimited in this way. With these commands, you can

- erase, delete, or display portions of text;
- create whitespace of a certain size;
- convert letters in portions of text to upper case or lower case;
- reformat text by squashing out white space and justifying text with right margin;
- copy or transfer parts of your text to another buffer.

Most of these commands take two arguments. *Li_exp* represents a relative offset from the current line. *Col_exp* represents a relative offset from the current column. If the /BLOCK switch is used, both line and column arguments are required. Without /BLOCK, *li_exp* and *col_exp* are optional.

The two commands that transfer or copy text to another buffer also require *bu_exp* as an argument. This is a buffer name from 0 to 9 or an expression that resolves to a buffer name.

The first line mark is at the current line and column. The second line mark is at the first mark plus *li_exp* lines and *col_exp* columns.

If *col_exp* is not specified, the second mark is at column 1. If no argument is specified, the delimited text extends from column 1 of the current line to column 1 of the next line. If you use the /BLOCK switch without specifying line and column arguments, you get an error message.

Delimited text strings start at the initial mark and go up to but do not include the end character position. (The end position can be marked before the beginning position.) Delimited text blocks include any two diagonally opposing corner positions. If the /BLOCK switch is not specified, the delimited text is treated as a text string.

These operations are generally easier to perform than it may seem at this point. When you are not specifying *col_exp*, you can usually calculate very rapidly the *li_exp* argument simply by looking at the text displayed. For example, if you want to delete a paragraph, simply move to the beginning of the paragraph and count the number of lines in the paragraph (including the current line). If the paragraph has 5 lines, your command reads simply:

```
> DELETE 5 ⮐
```

Table 3.3 describes the commands that operate on delimited text.

Command	Function
ERASE *[li_exp[,col_exp]]* ERASE[/BLOCK] *li_exp,col_exp*	Erases delimited text string (or block, if /BLOCK switch is used), leaving whitespace.
DELETE *[li_exp[,col_exp]]* DELETE[/BLOCK] *li_exp,col_exp*	Deletes delimited text string (or block, if /BLOCK switch is used). If text string is deleted, text is closed up. If text block is deleted, text to right overlays deleted block.
WHITESPACE *[li_exp[,col_exp]]* WHITESPACE[/BLOCK] *li_exp,col_exp*	Inserts whitespace the size of delimited string (or block, if /BLOCK switch is used). If text string is marked, delimited material and following text moves down. If block is marked, delimited material and any text to right of it moves to right.
TYPE *[li_exp[,col_exp]]*	Displays delimited text on screen. If no arguments are given, current line is displayed. Since display causes screen to scroll, issues another command-mode prompt to allow reading.
LOWERCASE *[li_exp[,col_exp]]* LOWERCASE[/BLOCK] *li_exp,col_exp*	Converts letters in delimited text string (or block, if /BLOCK switch is used) to lower case.
UPPERCASE *[li_exp[,col_exp]]* UPPERCASE[/BLOCK] *li_exp,col_exp*	Converts letters in delimited text string (or block, if /BLOCK switch is used) to upper case.
SQUASH *[li_exp[,col_exp]]*	Squashes extra whitespace out of delimited text, changing multiple spaces and tabs to single spaces (double spaces at end of sentence). If no argument is specified, only current line is affected.
JUSTIFY *[li_exp[,col_exp]]*	Same as SQUASH, except extra spaces are inserted between words where necessary so that affected lines are justified with right margin. A line will not be justified if more than three extra spaces per word are required.
COPY *bu_exp [li_exp[,col_exp]]* COPY[/BLOCK] *bu_exp li_exp,col_exp*	Copies delimited text string (or block, if /BLOCK switch is used) to buffer specified by *bu_exp*, destroying any previous contents. If specified buffer has open files, no text is copied, and an error message is displayed, as follows: Error: Buffer has open file(s)
TRANSFER *bu_exp [li_exp[,col_exp]]* TRANSFER[/BLOCK] *bu_exp li_exp,col_exp*	Same as COPY command, except transferred text is deleted from current buffer.

Table 3.3 Operations on delimited text

NOTE: *Remember that, if the /BLOCK switch is used, both line and column arguments are required.*

Formatting Text

Two of the commands listed in Table 3.3 require more explanation than is given above or in the table itself. These are the SQUASH and JUSTIFY commands, which allow you to reformat your text. The SQUASH command "squashes" extra whitespace out of delimited text, changing multiple spaces and tabs to single spaces. The JUSTIFY command is the same as SQUASH, except extra spaces are inserted between words where necessary so that affected lines are justified with the right margin.

Both of these commands work on paragraphs or units of text resembling paragraphs in that they are separated by one or more blank lines or a form feed. Any indentation of both the first and second lines of paragraphs marked for squashing are maintained. Subsequent lines line up with the second line. If the first paragraph of marked text has no second line, then the indentation of the first previous paragraph with a second line is maintained for the rest of the squashing operation. If there is no previous paragraph, the second and subsequent lines of squashed paragraphs line up with the left margin.

Squashed text is confined to the space between the left side of the squash area (as defined in the paragraph above) and the right margin.

Inserting/Overlaying Text

As Table 3.3 indicates, the COPY and TRANSFER commands place a text block or string in another buffer. You can later insert or overlay this text into the current buffer, using two of the commands described in Table 3.4. As this table shows, you can also insert text strings into the current buffer by means of the INSERT command.

NOTE: *If a buffer to which you have copied or transferred a block of text becomes the current buffer, the text loses its block attribute.*

Command	Function
INSERT */text_string/*	Inserts *text_string* at current position, sliding existing text right to prevent overwriting. String must be bounded by a character that does not occur within the string (arbitrarily shown here as a slash). Space, Tab, New Line and Form Feed cannot be used as string boundaries.
BINSERT[/BLOCK] *bu_exp*	Inserts text string or block from specified buffer into current buffer at current position. If text string is inserted, existing text is moved down. If text block is inserted, existing text moves to right. Inserted text may not come from source buffer containing files opened with any form of OPENFILE command.
BOVERLAY[/BLOCK] *bu_exp*	Inserts text string or block from specified buffer into current buffer at current position, overwriting existing text. Inserted text may not come from source buffer containing files opened with any form of OPENFILE command.

Table 3.4 Inserting/overlaying text

Searching and Changing Text Strings

The SEARCH and CHANGE commands enable you to locate easily a specific text string and change it if you wish. Several switches and templates are available to make these operations both flexible and specific.

The SEARCH command is similar in many respects to the FIND utility, which is described in Appendix A. This utility is especially useful when you are working at a low baud rate and displays of entire texts are impractical.

SEARCH Command

The SEARCH command enables you to locate any specified text string, wherever it appears in the text buffer. You execute the command from command mode. The syntax is:

> SEARCH*[/switch...] /search_string/* ⤶

The *search_string* must be bounded by a character that does not occur within the string (arbitrarily shown in the command line above as a slash). Space, Tab, New Line and Form Feed cannot be used as string delimiters.

A search always starts at the current position and ends when the text string is found or when the search is unsuccessful. If a forward search is successful, the new current position follows the last character in the string located; if a backward search is successful, the new current position is the first character of the string. If the search fails, the current position remains unchanged, and the following error message is displayed:

```
Unsuccessful search
```

To search for the same string another time requires issuing the command again. (Remember, however, that you can recover the previous command with CTRL-A.)

Also, if *search_string* is a null string (SEARCH //), then a search is conducted for a match of the last previous search string. Only the first 80 characters are remembered. If there was no previous search string, or if the string was longer than 80 characters, a warning message is displayed:

```
Warning: Incomplete string in search buffer.
```

CHANGE Command

The CHANGE command works like the SEARCH command, except the search string, when found, is deleted and replaced by a specified insert string. The syntax is:

> CHANGE*[/switch...] /search_string/insert_string/* ⤶

Again, both the search and insert strings must be bounded by a character that does not appear in the string. Note that the same character that bounds the two strings also serves as a separator between them.

After a change operation, SLATE automatically adjusts the spacing, taking into account any difference in length between the search string and the insert string.

Search/Change Switches

The SEARCH and CHANGE commands accept the same switches, with one exception. Table 3.5 describes all the switches that can be used with either or both of these commands.

Switch	Effect
/BACKWARD	Directs search from current position backward through the text buffer. If unsuccessful, search terminates at current position. If successful, search terminates at first character of search or insert string (new current position). By default, search is conducted forward from current position and terminates after last character of successful search.
/NCI	Directs SLATE to distinguish between upper- and lower-case letters in conducting search. By default, search is case insensitive. This switch makes it *not case insensitive* (NCI).
/WSP	Directs SLATE to match any whitespace character (space or tiny tab) or group of whitespace characters in search string with any number of whitespace characters in text buffer. By default, one whitespace character in search string matches only one such character in buffer.
/DO=*n*	Used with SEARCH command, searches for a string a specified number of times in order to find the *n*th occurrence. If search fails to find the *n*th or a previous occurrence of string, command is aborted and new current position becomes point at which search failed (i.e., after last character of previous successful search or, if /BACKWARD switch is used, at first character of previous successful search). Used with CHANGE command, changes search string to insert string specified number of occurrences, starting with next occurrence (or previous occurrence, if /BACKWARD is used).
/LAST	Used only with SEARCH command. Searches for last occurrence of search string in buffer. New current position is point at which search failed (i.e., after last character of previous successful search or, if /BACKWARD switch is used, at first character of previous successful search).
/ALL	Used only with CHANGE command. From current position, changes all occurrences of search string to insert string.
/TEMPLATES	Interprets following symbols as special SEARCH template characters. By default, these symbols are interpreted as normal text. ( ) { } [ ] | , - + * ~ # See Table 3.6 for explanation of templates.

Table 3.5 Switches for SEARCH and CHANGE Commands

Search/Change Templates

Templates are units of a search string that make search and change operations more flexible. These units consist of special template characters that perform specific functions (see /TEMPLATES, Table 3.5) and the part of the search string to be acted upon (the template expression).

Table 3.6 describes all the templates that can be used for search and change operations. Following are the definitions for a few terms used in the table.

Element	A character or range of characters. Ranges are indicated by two characters separated by a hyphen. For example, A, A-Z, and 0-9 are elements. In a range, the first character must be less than the second in the ASCII collating sequence. The range must be enclosed within square brackets.
Set	A group of elements listed in no particular order. The elements in a set are separated by commas and enclosed within square brackets.
Class	A character or set.
Template	An element, set, class, or template expression including one or more of the special template characters listed in Table 3.5.

NOTE: *Don't forget to use the /TEMPLATES switch with the SEARCH or CHANGE command when working with templates.*

Template	Function
|	Anchors the search string to the beginning or end of a line. For example, a matching string for /|A big dog/ must occur at the left margin. A matching string for /a big dog. |/ must occur at the right margin.
[1-3]	An example of an element in the form of a range of characters. Matches any occurrence of 1, 2, or 3.
[q,x,1-3]	An example of a set of elements. Matches any occurrence of q, x, 1, 2, or 3.
~ *class*	Matches any single character that does *not* match the class (character or set). For example, ~[a-z] matches any character that is not a letter.
{ *template* }	Matches either template or null string . For example, /big {brown }dog/ matches either "big brown dog" or "big dog."
{ *template* }*	Matches zero or more occurrences of the template. For example, /the {big }*shark/ matches "the shark," or "the big shark" or "the big big shark," etc.
{ *template* }+	Matches one or more occurrences of the template. For example, /the {big }+shark/ matches "the big shark," "the big big shark," etc. Note that /{big }+/ is equivalent to /big {big }*/.
(*tem1* | *tem2* |...*temN*)	Matches any single alternative within parentheses. For example, / (cat | dog) / matches either "cat" or "dog." A vertical bar (|) sets off the alternatives.
#	Matches any single character and thus useful in finding spelling or numeric variations. For example, /ind##es/ matches either "indices" or "indexes."

Table 3.6 Templates for search and change operations

XCLI Command

You can access a temporary (son) CLI without leaving SLATE. This capability is important, because otherwise several steps may be required to safely exit from SLATE and return to the CLI without losing work performed during the current session. By using the XCLI command, you can use the CLI - for example, to print or type a copy of a file you need for reference - without worrying about closing your current editing session.

There are two forms of command syntax for this command. The first form is

```
> XCLI ↲
```

The second form of the command is

> \> XCLI *CLI_command[;CLI_command;...]* ↲

With the first form, you enter the CLI and remain in it, receiving another CLI prompt after every command executed, until you terminate the program by typing BYE following the prompt. This returns you to command mode in SLATE. Pressing New Line again, in response to the command mode prompt, returns you to your previous position in page edit mode.

With the second form, you can list the CLI commands you want executed on the command line following the XCLI command, using semicolons as command separators. Pressing New Line executes the CLI commands on the command line and returns you to command mode in SLATE.

NOTE: *If you issue a CHARACTERISTICS command under the son CLI, SLATE will not become aware of any alterations to device characteristics you have made. For instance, changing the number of characters per line or lines per page will not cause SLATE to modify its screen display parameters. Also, certain characteristics will be restored when SLATE is exited to the same state they were in when SLATE began execution.*

Key Definition Mode

4

In key definition mode, you can change the function of a key on the keyboard or condense a sequence of functions into a keystroke. The key definitions created to perform new functions are called *key macros* or simply *macros.*

You can create a macro for most keys on the keyboard, including function keys and control characters. You cannot redefine keys such as SHIFT, CTRL, REPT, etc. Nor can you redefine CTRL-F1 and CTRL-P. (CTRL-F1 is the function key you use to enter key definition mode from page edit mode. CTRL-P prevents programmed functions from executing when you only want to type them as part of a key definition line.)

You can save the macros you create in key definition files (with the KEYSAVE command) and use them again whenever you execute SLATE simply by invoking the /I=*pathname* switch on the command line (see Table 1.1 in Chapter 1).

This chapter describes how to create key macros to perform the functions you need and how to save them for future use.

Entering Key Definition Mode

You enter key definition mode from page edit mode by pressing the Define Key function key (CTRL-F1). The key definition sequence is as follows:

CTRL-F1 *key_char* [*key_def_chars*] CTRL-F1

where

CTRL-F1	is the Define Key function key,
key_char	is the key you want to redefine,
key_def_chars	is the new definition.

The maximum length of a definition is equal to the number of columns on the screen (normally 80). Up to 25 key macros can be operative at any one time.

When you press the Define Key function key, the following prompt appears at the bottom of the screen:

```
Define:
```

You then press the key for which you want to create a macro. When you type the key, for example, the @ symbol, the prompt expands as follows to identify the key and its current definition:

```
Define: @ as:
@
```

If you do not want to redefine the key, press Define Key again (to exit from key definition mode and return to page edit mode). Otherwise, overwrite the @ on the second line (the current definition) or erase it with CTRL-U or Delete Line. Then type the new definition and press Define Key.

Key Symbols

Each key has an identification symbol associated with it, which is used to define the key and store the definition in a key definition file. (See "Saving Key Definitions" for information on this file.) The symbol for one of the normal typing characters is the character itself. Thus, in the example above, the symbol for the @ key is @.

Normally, you use an unprogrammed function key, if one is available, to define a key macro. The symbol for a function key becomes evident when you press the key after entering key definition mode. As in the example above, the definition line identifies the key by its symbol and gives the current definition. For example, if you press Define Key followed by the function key F3 (Close Whitespace), the prompt is

```
Define: ^s as:
^s
```

The symbol for F3 is ^s (CTRL-Uparrow, lower-case s). Note that control characters are normally dimmed on the screen (unless you used the /REVERSEDIM switch on entering SLATE). The symbols for all the function keys are listed in Appendix B.

Defining Key Macros

To create key macros to perform command sequences, you can use text strings, control characters, function keys, and keyword commands as part of the key definition. In what follows, we have discussed separately these various tools. But this is simply for ease of presentation. Do not lose sight of the fact that they can all be combined to create a single key macro.

Text Strings in Macros

Suppose you want to create a macro for the F3 function key that will type out the sentence, "SLATE is a text editor." Simply press Define Key followed by F3. Then overwrite the current definition and type in the new definition sequence (in this case, the text you want typed out). The screen now displays:

```
Define: ^s as:
SLATE is a text editor.
```

Press Define Key and your new key macro is ready to be executed. Just press F3 when you want this sentence to be typed out.

Whenever you want to revert to the original definition for this key, just enter key definition mode again, press F3, erase the current definition, and exit from key definition mode.

***NOTE:** Any key definitions you create are lost as soon as you exit from SLATE, unless you save them with the KEYSAVE command. (See "Saving Key Definitions" later in this chapter.)*

Command Sequences

When defining a command sequence for a key macro, two control characters are particularly useful: CTRL-P and New Line (CTRL-J).

Normally, when you type a control key, function key, or command keyword, it executes. In order to include one of these keys or keywords in a key macro *without* executing it, you must precede it with CTRL-P.

You must use New Line (or CTRL-J) in a key macro to terminate a command or command sequence, just as if you were executing the sequence then and there. (You need not enter CTRL-P before entering New Line in a key definition.)

Control Characters in Macros

Suppose you want a key similar to the DEL key except that it will delete the character at the current position instead of to the left of the current position. The procedure is as follows:

1. Enter key definition mode (press Define Key), type key to be defined (let's assume function key SHIFT-F2: symbol ^b), and erase current definition.
2. Type CTRL-P followed by CTRL-X to move cursor right; then type CTRL-P followed by DEL.
3. Press Define Key again.

The new definition line shows

```
Define ^b as:
X•
```

X represents CTRL-X and ◦ represents the DEL key. CTRL-P does not show up on the screen, but it does appear in the key definition file.

For a somewhat more complex example, suppose you want to create a macro that will type out the following memo format at the current position, indenting and placing the cursor following the first word, as shown.

```
To: _
From:
Subject:
```

The procedure is as follows:

1. Enter key definition mode (press Define Key), type key to be defined (again, let's assume function key SHIFT-F2: symbol ^b), and erase current definition.
2. Type in spaces for the indentation, followed by the word *To:* followed by New Line twice (to double space before next item in the format). Repeat for the words *From:* and *Subject:*
3. After the word *Subject:* do not double space but type CTRL-P, CTRL-W four times (to move cursor up to proper line) and type CTRL-P, CTRL-Y four times to move cursor left to proper column.
4. Press Define Key again.

The new definition line appears on the screen as follows:

```
Define: ^b as:
  To:JJ    From:JJ    Subject:WWWWYYYY
```

The letters J, W, and Y represent New Line (CTRL-J), CTRL-W, and CTRL-Y. Again, the CTRL-P characters you must enter before each stroke of the cursor movement keys (CTRL-W and CTRL-Y) do not show up on the screen, but they do appear in the key definition file. Note that any spaces in the definition line are preserved when the macro is executed.

Function Keys in Macros

You can also use function keys to create key macros. For example, suppose you want to define a key similar to CTRL-K, except that, instead of erasing from the current position to the end of the line, you want it to erase from the beginning of the line up to the current position. Proceed as follows:

1. Press Define Key; then type key to be redefined (again, let's assume ^b), and erase current definition.
2. Press CTRL-P followed by Mark Text function key.
3. Press CTRL-P again followed by Carriage Return to position second mark.
4. Press CTRL-P followed by Mark Text key again.
5. Type the operation code (the letter E, designating "erase").
6. Type New Line to terminate the delimiting operation and press Define Key to terminate the macro and return to page edit mode.

The new definition line appears on the screen as follows:

```
Define: ^b as:
^tM^tEJ
```

The symbol ^t is the symbol for the Mark Text function key. The M represents Carriage Return, E is the operation code, and J represents New Line (CTRL-J). CTRL-P does not show up on the screen, but it does appear in the key definition file.

In a similar way, you could create a macro to convert the letters in a word to upper (or lower) case. Use the above procedure, but substitute CTRL-F for Carriage Return to position the second mark. (CTRL-F moves cursor to beginning of next word.) And substitute the operation code U (upper case) or L (lower case) for E. When using the macro, just make sure current position is at the beginning of the word whose case is to be changed.

Keyword Commands in Macros

You can use keyword commands in macros in much the same way as you use control characters and function keys. In key definition mode (as in page edit mode), you must press the Command function key before you can type in a command keyword. However, in key definition mode, you press CTRL-P first, since you don't actually want to enter command mode then and there.

You must also be careful to use New Line to terminate the command and permit you to expand your key definition with functions not performed in command mode. (When you use two commands in sequence, you can terminate the first by using a a command separator (;) instead of New Line. This is explained below.)

Suppose you want to create a key macro to check buffer status. The definition sequence is as follows:

1. Press Define Key; then type key to be redefined (^b) and erase current definition.
2. Type CTRL-P and then press Command function key.
3. Type BSTATUS.
4. Type New Line to terminate the command and press Define Key to terminate the macro and exit from key definition mode.

The new definition line appears on the screen as follows:

```
Define: ^b as:
^qBSTATUSJ
```

The symbol ^q represents the Command function key and J represents New Line (CTRL-J). CTRL-P does not show up on the screen, but it does appear in the key definition file.

You could expand the above macro so it checks not only the buffer status but the file status of the current buffer. To do this, simply replace Step 3 as follows:

3. Type BSTATUS;FILESTATUS

If you did not use the semicolon in Step 3, as expanded, SLATE would view the FILESTATUS command as an argument and display the error message:

```
Error: Illegal variable name.
```

Continuing the above example, suppose you want a New Line to be entered after the BSTATUS command, so that the file status will appear on a separate line when the key macro is executed. Step 3 would then be as follows:

3. Type, in order: BSTATUS New Line FILESTATUS

Note that, in this case, New Line does not cause you to "leave" command mode, thus requiring another sequence of CTRL-P, Command key before you can enter the next command keyword. This is because, when SLATE executes the BSTATUS command, it does not leave command mode but issues another prompt. (See "The Command Line," Chapter 3.)

Interactive Command Sequences

SLATE allows you to define a key without terminating the last command in the key definition. In this case, when you use the macro, SLATE executes it as far as the last command, and then remains in command mode, waiting on the command line for input. In this way, you can create macros that allow you to interact with SLATE.

If you want to create a macro to execute a sequence of commands, but one in the middle requires input from the user, you can define two keys: one for the commands up to the point where input is required, and one for the remainder of the sequence.

For example, referring again to the above macro, as expanded, suppose you want the macro to give you a chance to change the buffer after you read the buffer status but before you read the file status. You would split the macro in two parts, as shown below.

Part 1

1. Enter key definition mode, type key to be redefined, and erase current definition.
2. Type CTRL-P and then press Command function key.
3. Type, in order: BSTATUS New Line BUFFER.
4. Press Define Key to terminate the macro (without typing New Line).

Part 2

1. Enter key definition mode, type different key to be redefined, and erase current definition.
2. Type CTRL-P and press Command function key.
3. Type FILESTATUS.
4. Type New Line to terminate the command and press Define Key to exit from key definition mode.

When you execute the key macro for Part 1, SLATE displays the buffer status and then pauses on the next command line to allow you to insert a buffer number. Insert the buffer number, press New Line, and then execute the key macro for Part 2.

Saving Key Definitions

Any key definitons you create with SLATE are lost as soon as you exit from the editor unless you save them first. The KEYSAVE command is your mechanism for saving key macros.

KEYSAVE

To use KEYSAVE, simply enter Command mode and type the command line:

```
> KEYSAVE pathname ↲
```

where *pathname* identifies the file in which you want the key definitions to be stored.

You can save several key definitions in the same file. Each definition saved is simply appended to the file. You may find it useful to keep more than one macro file, with macros grouped according to their general function. However, you can invoke only one key definition file at a time.

You invoke the definitions in a definition file on executing SLATE by using the initialization switch /I=*pathname* (see Chapter 1, Table 1.1). For example, suppose your key definitions are kept in a file called *key_def_file* in your working directory. If you want to use these definitions during an editing session, enter SLATE by typing the command line:

```
) XEQ SLATE/I=key_def_file pathname ↲
```

You can then edit the file specified by *pathname*, using key macros you created earlier and saved in *key_def_file.* The files you save with KEYSAVE contain all the symbols of the commands and the text you entered when defining the key.

KEYCLEAR

At times, during an editing session, you may want to remove key definitions you have created. The KEYCLEAR command does this for you. Just enter command mode and type KEYCLEAR. Any keys you have defined or redefined then revert to their original definition.

Automated Editing

5

The chapter introduces you to the various operators and operands that can be used in expressions. It also introduces several pseudo-commands helpful for automated editing. They are:

- [+DO] loops, which repeat a command sequence a specified number of times or until terminated by another influence;
- If-then-else sequences, which channel command sequences in one direction or another, depending on a certain condition;
- The [+EXIT] command, which terminates either a loop or the entire command line;
- The [+ASCII] command, which converts the octal value of an ASCII character into the character it represents (useful in conjunction with variable commands).

The chapter also describes the variable commands (V0-V9), which allow you to establish the value of up to 10 variables. These variables are generally used for looping/counting operations.

Finally, the chapter describes how to save the command sequences you create in command files and how to invoke them on executing SLATE.

Expressions

As you have noted already, you can use a variety of expressions as arguments to SLATE commands. Expressions involving several operators and operands can be used with many of the pseudo-commands and the variable commands described in this chapter as well as with commands discussed previously. Some command keywords, themselves, may be used as valid operands.

Operators

Operators are of two kinds: arithmetic and Boolean.

You can use arithmetic operators to perform simple calculations or to evaluate numerical expressions.

Boolean operators are used to perform logical operations. A Boolean binary operation is carried out on the corresponding bits of the two operands and not on zero vs. non-zero operands. Therefore, (100 & 001) is 000 rather than 001.

Table 5.1 lists both the arithmetic and Boolean operators that can be used with SLATE. It also shows the priority of operators for evaluation purposes. Operators with equal priority (addition and subtraction, for example) are evaluated from left to right.

Operator	Function	Priority
*	Multiplication (unsigned)	1 (highest)
/	Division (unsigned)	1
+	Addition (unary plus)	2
-	Subtraction (unary minus)	2
^	Boolean NOT	3
&	Boolean AND	4
!	Boolean OR	5
%	Boolean Exclusive OR	5 (lowest)
( )	Alters order of precedence in which operators within an expression are resolved. (Operators within parentheses are resolved first.)	

Table 5.1 Operators and operator precedence

Operands

Several keyword operands in addition to 32-bit integers can be used in expressions in SLATE. These are listed in Table 5.2. Any operand in the table may be abbreviated to the fewest number of characters that uniquely defines it.

Operand	Meaning
"	Numeric value of following ASCII character. For example, "x is the value of x (170 octal).
V*n*	Value of variable *n*, where *n* is a number from 0 to 9
VASCII	Numeric value of the ASCII character at the current position
BUFFER	Current buffer number (0-9)
COLUMN	Current column number
LINE	Current line number
LMARGIN	Column number of current left margin
RMARGIN	Column number of current right margin
VFAIL	Failure flag for SEARCH or CHANGE command. Can be used as an argument to [+EXIT] pseudo-command. Value is 1 at start of a command line and after every unsuccessful search. Value is 0 after a successful search.

Table 5.2 Valid operands in SLATE expressions

Pseudo-Commands

Normally, commands are executed in the order in which they appear in the command line. Several of the pseudo-commands modify this order by allowing

iterated (loop) and conditional operations,
insertions into the command line,
exiting from the command when a certain condition is met.

The pseudo-commands are listed in Table 5.3. Note that all pseudo-commands begin with a plus sign and the entire command, including arguments, must be surrounded by brackets. These brackets must by typed into the command line (unlike the brackets we have shown heretofore to indicate optional arguments).

NOTE: *Some of the sample commands in the rest of this chapter are too long to fit on one line of the manual's page format. Bear in mind that any elements of a command line that appears between the command-mode prompt (>) and the line terminator (↲) must be entered on one line.*

Pseudo-Command	Function
[+EQUAL *expr_1,expr_2*]	Compares numeric value of *expr_1* to that of *expr_2* and if they are equal (i.e., if condition is true) executes following commands up to next matching [+ELSE] or [+END]. If values are not equal, these commands are not executed.
[+NEQUAL *expr_1,expr_2*]	Same as [+EQUAL] except condition is true if *expr_1* and *expr_2* are *not* equal.
[+LESS *expr_1,expr_2*]	Same as [+EQUAL] except condition is true if *expr_1* is less than *expr_2.*
[+GREATER *expr_1,expr_2*]	Same as [+EQUAL] except condition is true if *expr_1* is greater than *expr_2.*
[+ELSE]	Alters the sense of [+EQUAL], [+NEQUAL], [+LESS] and [+GREATER] so that, if condition is not true, commands following [+ELSE] and preceding [+END] are executed. Otherwise, the commands are not executed.
[+END]	Marks the end of a range of a [+EQUAL], [+NEQUAL], [+LESS] or [+GREATER] pseudo-command.
[+DO] [+DO, *expr_times*]	Marks the start of the range of a [+DO] loop and executes commands occurring within a [+DO] - [+DEND] pair each time the loop is executed. The value of *expr_times* determines how many times the loop is executed. If *expr_times* is not given, loop is repeated until terminated by an unsuccessful search, a [+EXIT] command, or a console interrupt. If *expr_times* is not greater than zero, loop is skipped. Loops may be nested 10 deep. Unsuccessful SEARCH and CHANGE commands inside a [+DO] loop will terminate the loop, but they will not report an error or abort the command line.
[+DEND]	Marks the end of the range of a [+DO] loop.
[+EXIT] [+EXIT, *cond_expr*]	Terminates a [+DO] loop or the entire command line providing *cond_expr* is not specified or is greater than zero. If *cond_expr* is not greater than zero, [+EXIT] is ignored.
[+ASCII, *char_expr*]	Converts the low-order 8 bits of *char_expr* to a character to be placed in the command line.

Table 5.3 Pseudo-commands

NOTE: Do not use the character sequence [+ in a SLATE command line unless it is part of a pseudo-command.

Loops and Conditionals

SLATE supports simple and conditional loops. Simple repetitions are accomplished by [+DO] loops, which allow you to repeat an operation a specified number of times or until stopped by the [+EXIT] command (or, in the case of the SEARCH and CHANGE commands, until stopped by an unsuccessful search).

The conditional controls ([+EQUAL], [+NEQUAL], [+LESS], [+GREATER], [+ELSE]) are the SLATE mechanism for if-then-else sequences. (IF a condition is true, THEN do one thing; if not, do something ELSE.) When used inside [+DO] loops, conditional controls provide for conditional looping - that is, the operation will be repeated *if* a certain condition is met. The variable commands (see below) allows looping with counters - that is, the operation will be repeated *until* a certain condition is met.

Loops may be nested 10 deep, but loops, including nested loops, must be completed on the same line. Otherwise, an error condition exists.

Following are some examples of the use of loops and conditionals, showing proper command-line syntax. For further examples, see below under "Variable Commands."

[+DO] Loops

Suppose you want to format a title for a weekly status report and underline it. Type the following command lines:

```
> Line 3; Column 21; Insert /Weekly Status Report/ ↲
> Line 4; Column 21; [+DO, 20] Insert /-/; [+DEND] ↲
```

When you execute this command sequence, the title "Weekly Status Report" appears on line 3, beginning at column 21. The underline (a row of hyphens on line 4) appears below it.

The following example is more complex. In two commands, it creates a tic-tac-toe diagram, involving (in the second command) a nested [+DO] loop inside the string argument for the INSERT command. Try it and see what happens.

The spaces in the command lines represent spaces of four characters each. Note that the octal value of New Line is 12. Note also the two different means employed to produce the matching top and bottom parts of the diagram.

```
> INSERT /[+DO,2]    |[+DEND][+ASCII,12]/ ↲
> [+DO,2] INSERT /[+DO,11]-[+DEND][+ASCII,12]    |    |
[+ASCII,12]/;[+DEND] ↲
```

Conditional Controls

Consider the following simple string written with conditional pseudo-ops:

```
> INSERT/on the[+NEQUAL,3,5] Mississippi[+ELSE] Tennessee[+END]
mud./ ↲
```

The string is inserted as: "on the Mississippi mud," since 3 and 5 are not equal. Normally, of course, your first argument would consist of a variable expression which, when resolved, might or might not equal the second argument.

In the above example, the conditional keyword [+NEQUAL] could be replaced with [+LESS] without altering the outcome, since 3 is not equal to 5 and 3 is also less than 5. Note that one [+END] serves both the conditional keyword [+NEQUAL] and [+ELSE].

Variable Commands

The variable commands (V0-V9) allow you to define values for as many as 10 different variables. Since a variable can be used as a counter, variables are generally used for looping operations that involve counting.

The syntax of the V0-V9 commands is

V*var_name* [expr] ⤶

where

V*var_name*	is the command
var_name	is a variable name (a digit from 0 to 9)
expr	is any expression

If *expr* is not specified, the current value of V*var_name* is displayed. If *expr* is specified, the command sets the variable V*var_name* to the value of *expr.* For example:

```
> V0 ⤶
8
> V0 (120/V0)+1; V0 ⤶
16
>
```

This command generates output and causes the screen to scroll. After executing the command, SLATE remains in command mode and issues another prompt. Type New Line to return to page edit mode.

You can use the variable commands to calculate. For example:

```
> V0 7*3+(10/5-3) ⤶
> V0
20
```

Remember the rules for evaluating operators (see "Operators," above).

Looping/Counting with Variables

To count, set the variable to zero and then place a counter command inside a [+DO] loop. The format of the counter command is

```
V0 V0+1
```

In this case, the value of V0 will increment by one with each loop. If you set the counter (V0) to V0+5, the value of V0 will increase by five with each loop. If you set it to V0-1, it will decrement by one with each loop.

Suppose you want to know how many times a search string is matched. Use the variable command as follows:

```
> V0 0; [+DO,10] SEARCH /SLATE/; V0 V0+1; [+DEND]; V0 ⤶
2
>
```

Note the placement of the counter in the command line. The variable V0 records the number of successful searches up to 10. If you use the [+DO] command without specifying a number, the search operation continues until a search is unsuccessful.

Consider, also, the following example:

```
> V0 "A; [+DO,26] INSERT /[+ASCII, V0]/; V0 V0+1; [+DEND] ⤸
```

Variable 0 is assigned the value of A. The [+ASCII] pseudo-command in the string insert converts this value back to its ASCII representation. The counter increments the value of V0 on each of 26 passes. The string insert, then, consists of the upper-case alphabet, displayed as follows:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Or, consider the following (note that the octal value of ASCII 0 is 60):

```
> V0 1; V1 10; V2 100; V3 1000; V4 10000 ⤸

> V9 0; [+DO,5] V[+ASCII,60+V9]; V9 V9+1; [+DEND] ⤸
```

In this case, the counter is used to display the values of variables 0 through 4 as follows:

```
1
10
100
1000
10000
>
```

Note that you can combine a variable value with the argument to the [+ASCII] pseudo-command to perform counting operations.

Command Files

Command files are files that store command sequences. The key definition files described in Chapter 4 are a form of command file. These files are formed by creating key macros and then storing them in a file with the KEYSAVE command.

Besides defining key macros to perform specific tasks, you can also develop macros by typing command sequences directly into a file. You can then invoke this file by using the /I=*pathname* switch when you enter SLATE. This automatically executes the command sequence specified by *pathname*. (See "Saving Key Definitions" in Chapter 4 for a complete description of the /I=*pathname* switch.)

Suppose you want to format a page in the following way when you begin an editing session:

left margin at column 5;
right margin at column 74;
tabs at columns 10, 15, 20, and 25.

Create a command file and proceed as follows:

1. Type CTRL-P and then press Command function key (F1).
2. Type in the following line:
   LMARGIN 5; RMARGIN 74;TABCLEAR; TABSET 10, 15, 20, 25.
3. Finish by typing New Line.

If you need more than one line or if you prefer to line up the commands one under the other, simply begin each line with CTRL-P followed by F1; eliminate the command separators (semicolons); and end each line with New Line.

Now, when you want to format a file in this way when you edit it, simply call up your command file when you enter SLATE, using the /I=*pathname* switch.

Remember, you can invoke only one macro file at a time with this switch. However, you can combine command and key definition files by manipulating them with the various file and buffer commands described in Chapter 3.

The Find Utility

A

FIND is a utility program that examines a text file for the occurrence of a pattern. You use FIND in place of searches with text editors; FIND is especially useful when you are working at a low baud rate and displays of entire texts are impractical.

To specify a pattern (text string), you may type the pattern exactly as it occurs, or use one of several templates that match a range of patterns.

When the pattern is detected, FIND prints the line in which the pattern occurs. By using command line switches, you can direct FIND to print other information, such as the line number of each line containing a match, the number of lines containing matches, and a specifiable number of lines following the match.

FIND either recognizes or ignores differences between upper and lower cases, according to your specification. By default it is indifferent to cases (upper and lower).

Starting and Stopping FIND

You start FIND by typing the command line explained below. To stop FIND before it has finished inspecting the entire text, type CTRL-C CTRL-A.

The format of the command line is

) XEQ FIND[*/switch*]...*filename pattern* ↵

where

/switch	is one of the switches listed in Table 1.1.
filename	is the name of the file to be searched.
pattern	is a text string or template. The text string may not contain characters that have special meaning to the CLI. See the discussion below for explanation of pattern templates.

Switch	Action
/L	Directs output to @LPT instead of @TTO.
/L = *filename*	Appends output to *filename.* Creates *filename* if it doesn't exist.
/NCI	Sets FIND to detect differences between upper- and lower-case letters. When this switch is not used, FIND is case-indifferent.
/C	Prints a count of the number of lines containing matches.
/N	Prints line numbers preceding each output line.
/S = *integer*	Prints each line containing a match, and the number of succeeding lines specified by *integer.*
/H	Prints a header containing the name of the searched file.

Table A.1 Find Switches

Definition of Terms

An *element* is a character or a range of characters. Ranges are indicated by two characters separated by a hyphen. For example, A and a-z are elements. In a range, the first character must be less than the second in the ASCII collating sequence.

A *set* is a group of elements in no particular order. *Sets* are indicated by a list of elements separated by commas and enclosed within square brackets.

Escaped characters are alphabetic characters preceded by an Escape character, which stands for the control (CTRL) function. The Escape character is generated by pressing the Escape key on the left of the keyboard, and is represented by a dollar sign.

A *class* is either a character, an escaped character, or a set.

Pattern Templates

Anchoring a Pattern to a Margin

The vertical bar (|) at the right or left of a pattern matches zero characters and therefore "anchors" the pattern to the respective margin.

For example,

```
| Daddy
```

matches

```
Daddy is home
```

but neither

```
My Daddy is home
```

nor

```
    Daddy is home
```

Patterns not containing a vertical bar will match any occurrence of the pattern. Using anchored patterns greatly speeds FIND's performance.

Alternative Patterns

To specify alternative patterns, list the patterns within parentheses and separated by vertical bars. Each line containing any of the patterns is printed. For example:

```
qu| art
```

matches *kumquat* or *artichoke.*

Pattern or Null String

Setting a pattern in curly brackets causes the pattern to match itself or the null string. A null string matches at any character position on the line. For example:

```
a{b}c
```

matches *abc* and *ac.*

One or More Pattern Occurrences

Setting a pattern in curly brackets followed by a plus sign causes the pattern to match one or more occurrences of the pattern. For example:

```
a{bc}+
```

matches *abc, abcbc, abcbcbc,* etc.

Zero or More Pattern Occurrences

Setting a pattern in curly brackets followed by an asterisk causes the pattern to match zero or more occurrences of the pattern. Thus {*pattern*} + is equivalent to *pattern*{*pattern*}*.

For example, the set:

```
[a-c, k, x-z]
```

matches *a* or *b* or *c* or *k* or *x* or *y* or *z.*

Characters Not Members of a Class

Placing a tilde (~) in front of a class specification causes FIND to match any line that does not contain a member of the class.

For example:

```
~$1
```

matches any line that does not contain a tab (CTRL-I), and

```
~[a-z]
```

matches any line that does not contain any letters.

Any Single Character

The number sign (#) matches any single character. For example,

```
jain#ba
```

matches *jaineba, jainbba,* and *Jainaba*.

Function Key Symbols

B

Table B.1 shows the identification symbols for the function keys for the DASHER D2 (model 6053), D200 (models 6108 and 6109), D400 (model 6130) and D450 (model 6134).

The layout of the function keys on the keyboard is similar for all models except the D2. One template for the D2 (part no. 069-400047) and another for the other models (part no. 069-400046) are available. You can attach the one that most closely fits your CRT terminal above the function keys.

Function Key	Normal	Shifted	Control	Control Shifted
F1	q	a	1	!
F2	r	b	2	"
F3	s	c	3	#
F4	t	d	4	$
F5	u	e	5	%
F6	v	f	6	&
F7	w	g	7	' (apostrophe)
F8	x	h	8	(
F9	y	i	9	)
F10	z	j	:	*
F11	{	k	;	+
F12	|	l	<	, (comma)
F13	}	m	=	- (hyphen)
F14	~	n	>	. (period)
F15	p	` (accent grave)	0	(space)
C1	\	X	CTRL-\	CTRL-X
C2	]	Y	CTRL-]	CTRL-Y
C3	^	Z	CTRL-^	CTRL-Z
C4	_ (underscore)	[	CTRL-_	CTRL-[(Escape)

Table B.1 Function key identification symbols

NOTE: *The characters listed in the table are preceded by the function key header (CTRL-^). Function keys from F12 onward do not appear on DASHER D2 keyboard.*

Index

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Austraiia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogata
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

DataGeneral

Ordering Technical Publications

TIPS is the Technical Information and Publications Service—a new support system for DGC customers that makes ordering technical manuals simple and fast. Simple, because TIPS is a central supplier of literature about DGC products. And fast, because TIPS specializes in handling publications.

TIPS was designed by DG's Educational Services people to follow through on your order as soon as it's received. To offer discounts on bulk orders. To let you choose the method of shipment you prefer. And to deliver within a schedule you can live with.

How to Get in Touch with TIPS

Contact your local DGC education center for brochures, prices, and order forms. Or get in touch with a TIPS administrator directly by calling (617) 366-8911, extension 4086, or writing to

Data General Corporation
Attn: Educational Services, TIPS Administrator
MS F019
4400 Computer Drive
Westborough, MA 01580

TIPS. For the technical manuals you need, when you need them.

DGC Education Centers

Boston Education Center
Route 9
Southboro, Massachusetts 01772
(617) 485-7270

Washington, D.C. Education Center
7927 Jones Branch Drive, Suite 200
McLean, Virginia 22102
(703) 827-9666

Atlanta Education Center
6855 Jimmy Carter Boulevard, Suite 1790
Norcross, Georgia 30071
(404) 448-9224

Los Angeles Education Center
5250 West Century Boulevard
Los Angeles, California 90045
(213) 670-4011

Chicago Education Center
703 West Algonquin Road
Arlington Heights, Illinois 60005
(312) 364-3045

CUT ALONG DOTTED LINE

Technical Products Publications Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: ____________________

Document No. 069-400209-00

Yes	No			
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.	○ Other:
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ As a reference ○ As an introduction to the product	○ To instruct a class. ○ Other:
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ Other:	
☐	☐	Does the manual tell you all you need to know? What additional information would you like?		
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)		

Name: ____________________ Title: ____________________

Company: ____________________ Division: ____________________

Address: ____________________ City: ____________________

State: __________ Zip: __________ Telephone: __________ Date: __________

DG-06895

Data General

Data General Corporation, Westboro, Massachusetts 01580

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Technical Products Publications (C-138)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

Data General

users group

Installation Membership Form

Name ______________________ Position ______________________ Date __________

Company, Organization or School ____

Address ______________________ City ______________ State ______ Zip __________

Telephone: Area Code __________ No. ______________ Ext. ______________

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government
- ☐ Educational

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
COMMERCIAL ECLIPSE		
SCIENTIFIC ECLIPSE		
AP/130		
CS Series		
Mapped NOVA		
Unmapped NOVA		
microNOVA		
Other (Specify) ______		
______		
______		

3. Software

- ☐ AOS
- ☐ RDOS
- ☐ DOS
- ☐ Other
- ☐ MP/OS

Specify ______________

____

4. Languages

- ☐ Algol
- ☐ Assembler
- ☐ DG/L
- ☐ Fortran
- ☐ Cobol
- ☐ RPG II
- ☐ PASCAL
- ☐ PL/1
- ☐ Business BASIC
- ☐ Other
- ☐ BASIC

Specify ______________

____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ HASP
- ☐ CAM
- ☐ RJE80
- ☐ XODIAC
- ☐ RCX 70
- ☐ Other

Specify ______________

____

7. Application Description

○ ______________

____

____

____

____

____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify ______________

____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ ______________

____

Data General

Data General Corporation, Westboro, Massachusetts 01580. (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

Data General Corporation, Westboro, Massachusetts 01580

069-400209-0