



*EMACS Reference Guide*

*DOC5026-2LA*

---

# EMACS Reference Guide

*Second Edition*

Marion Shepp

*This guide documents the software operation of the  
Prime Computer and its supporting systems and utilities as  
implemented at Master Disk Revision Level 21.0 (Rev. 21.0).*

Prime Computer, Inc., Prime Park, Natick, MA 01760

## **COPYRIGHT INFORMATION**

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer, Inc. Prime Computer, Inc. assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1987 by Prime Computer, Inc., Prime Park, Natick, Massachusetts 01760

PRIME, PRIME, PRIMOS, and the PRIME logo are registered trademarks of Prime Computer, Inc. DISCOVER, EDMS, FM+, INFO/BASIC, INFORM, Prime INFORMATION, Prime INFORMATION CONNECTION, MDL, MIDAS, MIDASPLUS, PRIME MEDUSA, PERFORM, PERFORMER, PRIME/SNA, PRIME TIMER, PRIMECALC, PRIMELINK, PRIMENET, PRIMEWAY, PRIMEWORD, PRIMIX, PRISAM, PRODUCER, Prime INFORMATION/pc, PST 100, PT25, PT45, PT65, PT200, PW153, PW200, PW250, RINGNET, SIMPLE, 50 Series, 400, 750, 850, 2250, 2350, 2450, 2550, 2650, 2655, 2755, 6350, 6550, 9650, 9655, 9750, 9755, 9950, 9955, and 9955II are trademarks of Prime Computer, Inc.

Ann Arbor is a trademark of Ann Arbor Terminals, Inc.

Concept is a trademark of Human Designed Systems, Inc.

## **PRINTING HISTORY**

First Edition (IDR5026) April 1982 for Release 18.3

Update 1 (PTU2600-105) May 1983 for Release 19.2

Update 2 (PTU2600-107) May 1984 for Release 19.3

Update 3 (UPD5026-13A) January 1986 for Release 20.0

Second Edition (DOC5026-2LA) January 1988 for Release 21.0

## **CREDITS**

*Design:* Leo Maldonado

*Editorial:* Margaret Hill

*Document Preparation:* Anne Marie Fantasia, Celeste Henry, and Kathy Normington

*Production:* Judy Gordon

*Composition:* Julie Cyphers, Anne Marie Fantasia

## HOW TO ORDER TECHNICAL DOCUMENTS

Follow the instructions below to obtain a catalog, a price list, and information on placing orders.

*United States Only:* Call Prime Telemarketing, toll free, at 800-343-2533, Monday through Friday, 8:30 a.m. to 5:00 p.m. (EST).

*International:* Contact your local Prime subsidiary or distributor.

## CUSTOMER SUPPORT CENTER

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (within Massachusetts)

1-800-541-8888 (within Alaska)

1-800-651-1313 (within Hawaii)

1-800-343-2320 (within other states)

## SURVEYS AND CORRESPONDENCE

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department  
Prime Computer, Inc.  
500 Old Connecticut Path  
Framingham, MA 01701

---

# Contents

<b>About This Book</b>	ix
<b>1 Getting Started With EMACS</b>	1-1
Introduction	1-1
Common EMACS Terms	1-1
EMACS Command Conventions	1-3
Organization of the EMACS Screen	1-6
Entering the EMACS Environment	1-9
Saving Your Work	1-12
Exiting From EMACS	1-13
Basic EMACS Commands	1-14
<b>2 Summary of EMACS Commands by Function</b>	2-1
Introduction	2-1
Help Commands	2-2
Cursor Movement Commands	2-3
Screen Display Commands	2-4
Editing Commands	2-5
File Management Commands	2-15
Buffer and Window Commands	2-16
Macros	2-17
PRIMOS Command Execution	2-19
General Modes	2-19
Environment Commands	2-23
Information Commands	2-24
Commands for Slow Terminals	2-25
Library Commands	2-25
Speed-type Commands	2-25
Miscellaneous Commands	2-26
<b>3 Dictionary of EMACS Commands</b>	3-1
Introduction	3-1
Dictionary of Commands	3-1

<b>4</b>	<b>Online Help Facility</b>	4-1
	Introduction	4-1
	EMACS Help Command	4-1
	Apropos	4-2
	Explain_Key	4-2
	Describe	4-3
	L Option	4-4
	? Option	4-5
<b>5</b>	<b>Speed-type</b>	5-1
	Introduction	5-1
	How Speed-type Works	5-1
	Speed-type Commands Reference Section	5-13
<b>6</b>	<b>Customized Library Files</b>	6-1
	Introduction	6-1
	User Library Files	6-1
	File Hooks	6-3
	Creating Your Own Interface	6-6
	Simplifications You Can Make	6-7
<b>7</b>	<b>The TERMCAP Facility</b>	7-1
	Introduction	7-1
	Using TERMCAP With EMACS	7-2
	TERMCAP Capabilities	7-4
	Adding an Entry to the TERMCAP Database	7-11
<b>8</b>	<b>Language Modes</b>	8-1
	Introduction	8-1
	COBOL Mode	8-1
	C Mode	8-7
	FORTRAN Mode	8-13
	RPG Mode	8-16
	LISP Mode	8-24
	Common LISP Mode	8-26
	Listener Mode	8-28
	Additional Information About Compiling and Debugging Programs	8-29

<b>A</b>	<b>Alphabetical Summary of Commands</b>	A-1
<b>B</b>	<b>Prime Extended Character Set</b>	B-1
	Specifying Prime ECS Characters	B-1
	<b>Index</b>	Index-1

---

# About This Book

## Purpose

The *EMACS Reference Guide* is a reference source for the EMACS editor. It is intended to provide the user with detailed information about any specific EMACS command. The central focus of this new edition is the listing and description of all current EMACS commands in Chapters 2 and 3.

Chapter 1 provides instructions for new users about starting and running EMACS. In addition, the book contains information about how to use EMACS' online help facility, the speed-type abbreviation facility, TERMCAP, and EMACS' high-level language modes.

## Organization

The *EMACS Reference Guide* contains eight chapters and two appendices, which are summarized below.

### Chapter 1 — Getting Started With EMACS

This chapter introduces procedures for starting up and running EMACS. The chapter includes a list of ten basic EMACS commands.

### Chapter 2 — Summary of EMACS Commands by Function

This chapter lists all of the current EMACS commands, by function. It contains a discussion of each functional category and lists associated command names with keybindings (if any) and short definitions.

### Chapter 3 — Dictionary of EMACS Commands

This chapter contains a description of every standard EMACS command. The commands are listed alphabetically, by command name.

### Chapter 4 — Online Help Facility

This chapter describes the EMACS online help commands. The purpose of the chapter is to encourage you to use this useful, but little-known online help facility of EMACS.

### Chapter 5 — Speed-type

This chapter provides instructions for using the EMACS abbreviation facility. The speed-type environment enables you to define an abbreviation for a single word or a large amount of text. When you type the abbreviation, EMACS expands it automatically, thus saving you time and keystrokes.

### Chapter 6 — Customized Library Files

This chapter explains how to construct a startup library file so that you can set up customized parameters when you invoke EMACS.

### Chapter 7 — The TERMCAP Facility

This chapter is for experienced users who want to set up their own terminal configurations or use EMACS on a non-Prime terminal. It contains instructions for using the TERMCAP descriptions of terminal capabilities, and writing a TERMCAP entry.

### Chapter 8 — Language Modes

This chapter explains how to use EMACS programming language modes to enter, edit, compile, and debug programs in the high-level languages that Prime EMACS supports.

### Appendix A — Alphabetical Summary of Commands

This appendix lists all current EMACS commands by keybinding with a short description of what each command does. Extended commands are listed by command name.

### Appendix B — Prime Extended Character Set

This appendix discusses the expanded ASCII character set that becomes effective at Rev. 21.0.

## Related Documentation

These documents provide related information:

- *EMACS Extension Writing Guide* (DOC5025-2LA)
- *EMACS Primer* (IDR6107-183P)
- *EMACS Standard User Interface Guide* (DOC7446-2LA)
- *EMACS Reference Card* (IDR5026-1RA)
- *Prime User's Guide* (DOC4130-4LA, UPD4130-41A, UPD4130-42A)

Online tutorials for fundamental mode are located in EMACS\*. The tutorials are:

- TEACH-EMACS-FUND-INSTRUCTIONS
- TEACH-EMACS-FUND-1
- TEACH-EMACS-FUND-2
- TEACH-EMACS-FUND-3
- TEACH-EMACS-FUND-4

## Acknowledgments

The author wishes to thank Mary Hadcock, Tom Bugos, Peter Neilson, John Seybold, Jerry Ornstein, Matt Carr, George Gove, and all others who contributed to and reviewed this document.

## Prime Documentation Conventions

The following conventions are used in command formats, statement formats, and in examples throughout this document. Examples illustrate the uses of these commands and statements in typical applications.

<i>Convention</i>	<i>Explanation</i>	<i>Example</i>
<b>lowercase boldface</b>	In text, words in lowercase bold-face indicate the names of commands, functions, and variables.	<b>next_buf</b>
<i>italic</i>	In command formats, words in italic indicate variables for which you must substitute a suitable value.	<b>-SPDT</b> <i>pathname</i>
<b>Boldface italic</b>	In examples, user input is bold-face italic but system prompts and output are not.	Command: <b><i>save_all_files</i></b>
Brackets [ ]	Brackets enclose a list of one or more optional items. Choose none, one, or more of these items.	LD [-BRIEF -SIZE]
Braces { }	Braces enclose a list of items. Choose one and only one of these items.	CLOSE {filename ALL}
Vertical bars 	Vertical bars enclose a list of two or more options. Choose one or more of these items.	OUTPUT   filename     TTY
Ellipsis ...	An ellipsis indicates that the preceding item may be entered more than once on the command line.	SHUTDOWN pdev-1 [...pdev-n]
Parentheses ( )	In command or statement formats, you must enter parentheses exactly as shown.	DIM array (row, col)

<i>Convention</i>	<i>Explanation</i>	<i>Example</i>
Hyphen -	Wherever a hyphen appears as the first character of an option, it is a required part of that option.	SPOOL -LIST
Key symbol	In examples, the name of a key enclosed within a rectangle indicates that you press that key.	Press <input type="text" value="Return"/>
Angle brackets in messages < >	In messages, a word or words enclosed within angle brackets indicates a variable for which the program substitutes the appropriate value.	Disk <diskname>

---

# 1 Getting Started With EMACS

## Introduction

This chapter contains essential information for using the EMACS full-screen editor. It discusses the following topics:

- Common EMACS terms
- EMACS command conventions
- Organization of the EMACS screen
- Entering the EMACS environment
- Saving your work
- Exiting from EMACS
- Basic EMACS commands

## Common EMACS Terms

### *buffer*

A work space in which EMACS contains the text that you create or edit.

### *command*

A directive that specifies the operation to be performed. Most EMACS commands can be executed by typing an associated character sequence. All commands can be executed by typing `[Esc] [X]`, followed by the command name.

### *EMACS Standard User Interface (SUI)*

A version of the EMACS editor that assigns commands to function keys on the terminal keyboard.

### *EMACS Standard User Interface with Extensions (SUIX)*

A version of the EMACS editor that extends the EMACS SUI by incorporating both the bound function keys and fundamental mode commands.

***function***

One or more EMACS programming statements designed to accomplish a specific task. Functions may be compiled and executed from the current buffer or they may be executed by entering the function name at the **PL:** prompt.

***fundamental mode***

The default mode for EMACS. All commands are prefaced by either the Escape or the Control key.

***keybinding***

The association of a keypath with a command.

***keypath***

The keystroke sequence that causes a command to execute. A keypath may have ten keystrokes.

***macro***

A group of EMACS commands that is constructed and saved so that it can be executed as a single command.

***minibuffer***

The second and third lines of the four lines at the bottom of your screen. EMACS uses the minibuffer to issue prompts and messages that give instructions to the user.

***mode***

A method of operation. In EMACS, the mode determines which keybindings are in effect for the EMACS commands.

***prefix***

A character sequence that precedes a command in EMACS. The most common prefix keys are the Escape key and the Control key.

***status line***

The first line of the four lines at the bottom of your screen. The status line gives you information about the current version of EMACS, the mode you are in, the name of the current buffer, whether you have modified the buffer, and the pathname of your current work file.

***window***

A portion of the screen, after it has been divided horizontally or vertically. The sections are separated by a line of dashes (- or |).

## EMACS Command Conventions

When you first invoke EMACS, you are placed, by default, in a working environment known as *fundamental mode*. This environment contains a basic set of EMACS *commands*, which are available for use at any time. All the EMACS commands in fundamental mode begin with either the Control key (`Ctrl`) or the Escape key (`Esc`). The Control key works like a shift key in that you hold it down while you are typing the next character; you do not hold down the Escape key. The following examples show how this convention is used in this book:

- `Ctrl X` (where *x* is any character) indicates that you must press the `Ctrl` key and hold it down while you type the next character.
- `Esc xx` (where *xx* is any character or string of characters) indicates that pressing the key and typing the character(s) are two *separate* actions.

Whenever you press `Ctrl` along with another character or press `Esc` followed by another character or sequence of characters, EMACS interprets what you have typed as a command. Any characters you type that are not preceded by one of these keys are interpreted as text to be inserted into your document. To use EMACS successfully, you must have a terminal that has Escape and Control keys so that EMACS can distinguish commands from ordinary text.

Prime supports two enhancements to fundamental EMACS, the *EMACS Standard User Interface (SUI)* and the *EMACS Standard User Interface with Extensions (SUIX)*. Both of these interfaces enable a set of functions that are bound to the terminal's special function keys. The EMACS SUI is limited to these functions. The EMACS SUIX enables you to use these functions in addition to the fundamental mode commands. The *EMACS Standard User Interface Guide* is a guide for both the EMACS SUI and the EMACS SUIX.

### Note

Refer to the About This Book section for explanations of conventions used in this book to indicate function keys in command formats and text formats.

When you use the EMACS SUIX, SUIX command keybindings may overwrite fundamental mode command keybindings. For example, on the PT200™ terminal, `Esc N` cannot be used for the `next_buf` command because some of the SUIX function keys transmit character streams beginning with `Esc N`, causing the `Esc N` to be usurped by the function keybinding. In this case, the user would use `Esc n` (lowercase) for `next_buf`.

## Functions

All EMACS *functions* are Prime EMACS Extension Language (PEEL) statements that can be executed by typing `Esc Esc`. The `PL:` prompt appears in the *minibuffer* (see The Minibuffer section later in this chapter) at the bottom of your screen. Respond to the prompt by typing the function name and its arguments, enclosed in parentheses. An example of executing the `self_insert` function to produce an asterisk on your screen is shown below:

```
PL: (self_insert "*")
```

For more information about executing functions, see the *EMACS Extension Writing Guide*.

## Commands

EMACS commands are really Prime EMACS Extension Language (PEEL) functions that are bound to sequences of keystrokes. For example, the `[Ctrl N]` command, which moves the cursor to the next line, is bound to the function called `next_line_command`. When you type `[Ctrl N]`, you are invoking the `next_line_command` function.

In most cases, EMACS commands are bound to keys that relate to the name of the function. You do not need to know the names of these functions in order to use EMACS. However, as you begin to customize EMACS, the names of these functions will be important. For this reason, the dictionary section of this book (Chapter 3) lists the commands alphabetically by function name, which is the same as the command name. The function name is also a good description of what a command does. To help you identify a command name if you know only the keybinding, Appendix A of this book lists all the EMACS commands alphabetically by keybinding.

## Extended Commands

Other EMACS functions that are not bound to keystrokes are called *extended* commands. To invoke an extended command, you type `[Esc] [X]`. The **Command:** prompt appears in the minibuffer. Respond to the prompt by typing the name of the command (the function name) and pressing `[Return]`.

For example, to issue the `save_all_files` command, you type `[Esc] [X]`. When the **Command:** prompt appears in the minibuffer, type the following response:

```
Command: save_all_files
```

## Keybindings

A *keybinding* is a sequence of characters (*keypath*) that EMACS uses to invoke a command. For example, the keypath `[Ctrl D]` tells EMACS to invoke the `delete_char` command. Also, some EMACS fundamental mode commands are bound to character sequences for special keys on your terminal as well as to keypaths. These bindings vary depending on the type of terminal you are using. For instance, the `[Ctrl H]` command deletes the character immediately preceding the cursor. Most terminals have a Backspace key that does this as well.

## The Point

Most EMACS commands take effect before or after *point*, the location in your text where editing takes place. The current cursor position indicates the position of point, which lies between the character on which the cursor rests and the one immediately preceding it. Point lies between these two characters, and not on either one.

Figure 1-1 illustrates the location of point in a line of text. Note that blank spaces and line separators are considered characters even though they are not visible on the screen.



Figure 1-1  
Location of Point

## Giving Numeric Arguments to EMACS Commands

You can give a numeric argument to almost any EMACS command. In most cases, EMACS interprets a numeric argument as a repetition count for the command following it.

An argument of 1 is the default for most commands, and it does not usually need to be specified. Positive arguments repeat a command the specified number of times. Negative arguments repeat a command the specified number of times in the opposite direction. For example, the `Ctrl F` command moves the cursor forward one character without an argument. If you give `Ctrl F` an argument of 10, the cursor moves forward 10 characters. If you give the same command an argument of -10, the cursor moves backward 10 characters.

A few EMACS commands interpret certain arguments as command modifiers that change the way the command takes effect. These unusual cases exist for convenience and are documented as they come up. For example, the `Ctrl K` command without an argument "kills" all text on the current line. If you give `Ctrl K` an argument of 0, it kills all text from point back to the beginning of a line.

There are two ways to give numeric arguments to EMACS commands. The easiest way to specify an argument is to precede a command with the Escape key, followed by an optional minus sign and/or a digit or digits. For example,

<i>Command</i>	<i>Action</i>
<code>Esc 6 Ctrl F</code>	Moves the cursor forward six characters.
<code>Esc 2 Esc B</code>	Moves the cursor backward two words.
<code>Esc - 3 Ctrl Z</code>	Moves the cursor down three lines.
<code>Esc 1 0 *</code>	Inserts ten asterisks (*) into your text.

### Note

Normal text characters, (\* in the above example) act as text characters when they are preceded by `Esc n`.

The second way to specify an argument is by typing `Ctrl U`. The `Ctrl U` keystrokes have a special use in passing numeric arguments to EMACS commands. The placement of `Ctrl U`, directly preceding a command, or before or after a numeric argument, is strategic in determining its effect.

There are three ways to use `Ctrl U`:

1. Before a command
2. After a numeric argument
3. Before a numeric argument

Typing `Ctrl U` before a command usually means to multiply by 4. In effect, typing `Ctrl U` once passes the subsequent command an argument of 4; typing `Ctrl U` twice passes the command an argument of 16; three times, 64; and so on. For example,

<i>Command</i>	<i>Action</i>
<code>Ctrl U</code> <code>Esc</code> <code>D</code>	Deletes four words following point.
<code>Ctrl U</code> <code>-</code>	Enters four dashes (-) into your text.
<code>Ctrl U</code> <code>Ctrl U</code> <code>-</code>	Enters sixteen dashes (-) into your text.

#### Note

Normal text characters (- in the above example) act as text insert characters when they are preceded by `Ctrl U`.

Typing `Ctrl U` immediately after a numeric argument multiplies the numeric argument by 4. For example,

<i>Command</i>	<i>Action</i>
<code>Esc</code> <code>2</code> <code>Ctrl U</code> <code>-</code>	Enters eight dashes (-) into your text.

Placing `Ctrl U` before a numeric argument, followed by a command, specifies the argument to that command. In this case, `Ctrl U` functions the same way as `Esc`. For example,

<i>Command</i>	<i>Action</i>
<code>Ctrl U</code> <code>6</code> <code>Ctrl F</code>	Moves the cursor forward six characters.
<code>Ctrl U</code> <code>2</code> <code>Esc</code> <code>B</code>	Moves the cursor backward two words.

## Organization of the EMACS Screen

EMACS divides your screen into three areas, each one displaying different information. These are:

- The text area
- The EMACS status line
- The minibuffer

The terminal's own status line remains at the bottom of the screen.

Figure 1-2 illustrates a typical EMACS screen.

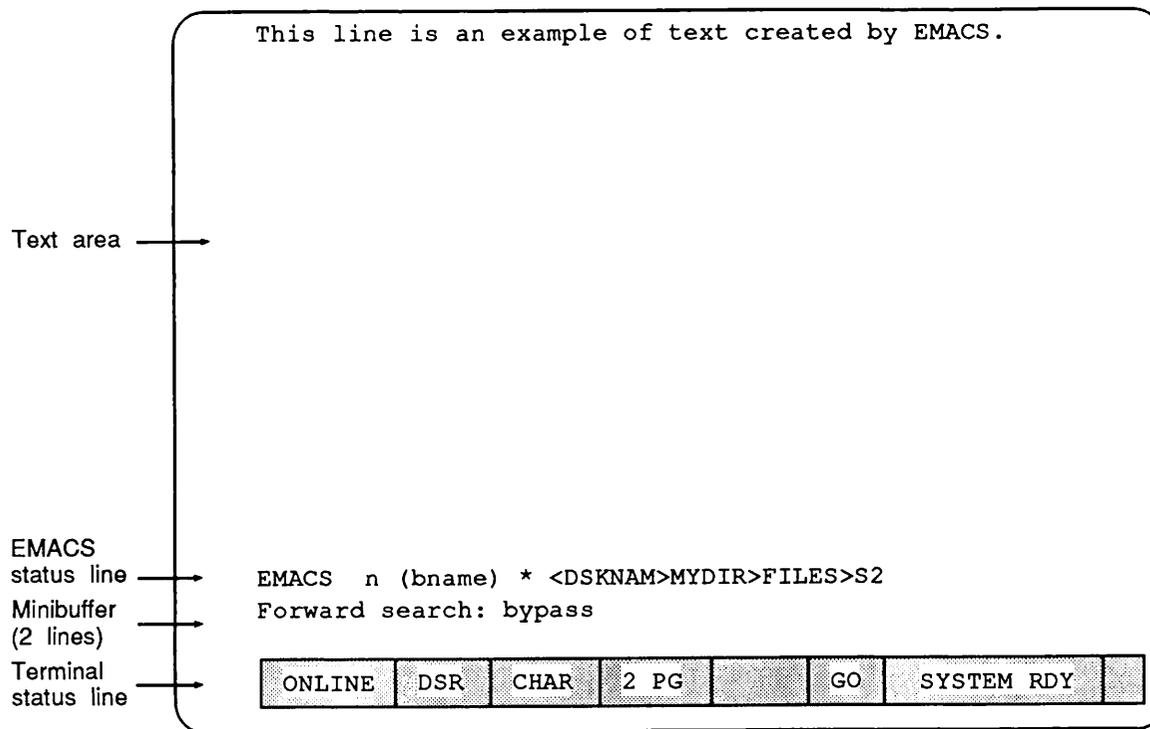


Figure 1-2  
A Typical EMACS Screen

## The Text Area

The text area is the top 21 lines of the EMACS screen. It contains the terminal's cursor and displays the text you are editing. All of your editing changes take place in this area.

**Messages in the Text Area:** The top lines of the text area are sometimes preempted by some information that is not part of your file. This information may be a result of certain EMACS commands or messages from the system. This information does not alter your file in any way. It disappears as soon as you type your next EMACS command.

**Line Numbers:** Text line numbers are not normally displayed in EMACS because EMACS is not limited to treating text by line number. It performs faster without the number display. To issue the line numbering command, type `[Esc] [X]`; when you see the **Command:** prompt in the minibuffer, type the following response:

Command: `#on`

The display changes to include a line number at the beginning of each line. Note that line numbers do *not* become part of your file, but are displayed for your information only. To remove line numbers from the display, type:

Command: **#off**

To learn whether line numbering is in effect, type the following:

Command: **#**

A message appears in the minibuffer telling you whether line numbers are on or off.

## The EMACS Status Line

The EMACS *status line* is the fourth line from the bottom of the screen. (The bottom line on some terminals gives *terminal* status information.) The status line displays useful information about the text you are currently editing. It shows you what is going on in EMACS and why your commands may not be interpreted in the standard way. If you are surprised by the way EMACS has reacted to your commands, look at the status line for assistance.

**Status Line Components:** This section describes the components of the status line shown in Figure 1-2. This line appears as follows in the display:

```
EMACS <n> <(bname)> * <DSKNAM>MYDIR>FILES>S2
```

where:

EMACS <n>	Indicates the version number of the EMACS editor being used.
<(bname)>	Is the name of the current <i>buffer</i> . (See Chapter 2.) EMACS can maintain several buffers in one editing session. Although you can edit only one buffer at a time, multiple buffers make it easy to work with a number of files. When you move from one buffer to another, it is helpful to have the name of the buffer in which you are currently working displayed on the status line. Normally, a buffer takes its name from the file it contains if the file is an existing file and has been loaded from a disk.
*	Indicates that you have made unsaved changes to the text in your current editing session. If a file has not been altered since it was read in or saved, no asterisk is present.
<DSKNAM>MYDIR>FILES>S2	Is a standard PRIMOS® <i>pathname</i> . It describes the tree structure leading to the file currently being edited. In this case, the filename is S2, and it is identical to the buffer name.

**Additional Information:** The status line may display other useful information as the need arises. For instance, if you type a line longer than 80 characters, the current terminal setting may not permit all the text to be displayed. To tell you what position you have reached that is not currently displayed, EMACS places the number of that column position at the far left of the status line as soon as the cursor goes off the screen edge.

## The Minibuffer

The two lines under the EMACS status line constitute the *minibuffer*. EMACS uses this area to print responses to certain commands, to give instructions to the user, and to issue error messages.

For example, a prompt appears in the minibuffer in response to the `Ctrl S` command. That prompt and a sample response are as follows:

```
Forward search: bypass
```

The next example shows the error message that appears in the minibuffer when you try to save a file that you have not named:

```
No default file name for this buffer
```

## Entering the EMACS Environment

To enter the EMACS environment, type the following standard command line after the PRIMOS `OK`, prompt. (The examples in this book assume that the `OK`, prompt is used.)

```
EMACS [options]
```

The order of options is irrelevant. A command line with options is shown below:

```
EMACS pathname -TTP terminal_type
```

The EMACS command invokes the EMACS editor. All of the arguments are optional, although you need to indicate the terminal type by one of the ways discussed below.

### Pathname

EMACS was designed to be able to edit text files (files of characters) consisting of lines, each terminated by a new line character. To edit an existing file, specify the pathname of that file on the EMACS command line. (If it is a file in your current directory, you need to specify only the simple filename.) EMACS then puts you into a buffer that has the same name as the specified file. The contents of the buffer are displayed on the screen.

If you are creating a new file, you may specify a new name in the pathname, or you may omit the pathname. In the latter case, EMACS puts you into a buffer called *main*. (See the Buffer and

Window Commands section in Chapter 2 for more information about buffers.) You are then free to enter text and to use all the EMACS facilities. If you decide to save the text in a new file, you name the new file at the time of the save.

#### Note

A valid PRIMOS filename may begin with a hyphen. If you specify a filename that begins with a hyphen as a command line option, and accidentally, the filename is identical to a valid EMACS option specifier, for example, `-SUIX`, the filename must be enclosed in single quotation marks. The following command line invokes SUIX mode and looks for a file named `-SUIX`.

```
EMACS '-SUIX' -TTP PT200 -SUIX
```

## Terminal Type

EMACS needs to know your terminal type. It checks the command line and the `.TERMINAL_TYPE$` global variable, looking for a terminal type definition. If it does not find a definition in either of these places, EMACS prompts you to enter a terminal type interactively. If you specify your terminal type as "unknown" or use an abbreviation that is not defined, EMACS responds with an error message and does not let you continue.

Chapter 7 contains information about `.TERMINAL_TYPE$` and the TERMCAP database, and also gives instructions for using EMACS with a non-Prime terminal.

Use the following format for the `-TTP` option on the command line:

```
-TTP terminal_type
```

Your terminal type can be one of the three Prime-supported terminals (the PST 100™, PT200, or PT45™) or it can be one of the terminals listed in the TERMCAP database, `EMACS*>TERM>TERMCAP`. (The PT200W is not a separate model. The term refers to a PT200 that has been set up via the SETUP menu to work in 132-column mode.)

## Other Options

When you initialize EMACS, you can also specify any of the following options:

<i>Option</i>	<i>Description</i>
<code>-ECHO_CPL</code>	Causes EMACS to perform screen updates while reading a CPL or COMINPUT file. This option is viable only if EMACS is invoked by CPL or COMINPUT.
<code>-HEIGHT</code>	Sets terminal display at <i>n</i> lines.
<code>-HELP</code>	Prints a list of available options.

- NOXOFF** Tells EMACS to treat `Ctrl S` and `Ctrl Q` as EMACS `forward_search` and `quote_command` commands instead of the PRIMOS stop and start print commands, which freeze and unfreeze the terminal display. Allows `Ctrl X` `Ctrl S` to be used as EMACS `save_file` command.
- NULIB** Tells EMACS not to look for either of the initialization files `EMACS*>INIT.EM` or `EMACS*>INIT.EFASL`. With neither the `-ULIB` nor the `-NULIB` option specified, EMACS searches first for `EMACS*>INIT.EM`, then for `EMACS*>INIT.EFASL`. The System Administrator can move or rename the `EMACS*>INIT.EM` file or you can use the `-ULIB` option to specify the `EMACS*>INIT.EFASL` file.
- SAVE\_SCREEN** Saves the contents of your screen as it was when you invoked EMACS. When you exit from EMACS, the cursor is restored to its original location as it was just before you entered EMACS. This option works only on the PT200 terminal in 48x80 display mode.
- SPDT *pathname*** Turns on speed-type, the EMACS abbreviation handling environment, and loads in the file of speed-type abbreviations named *pathname*. (See Chapter 5 for information about speed-type.)
- SPEED, *-bps*** Sets the terminal display algorithm to use bps. Default bps is 9600. A value from 0 to 32,767 will be accepted as specified. A value greater than 32,767 becomes the default (9600).
- SUI** Invokes the EMACS Standard User Interface.
- SUIX** Invokes the EMACS Standard User Interface with Extensions.
- ULIB *pathname*** Loads an EMACS library file and compiles it. (See the *EMACS Extension Writing Guide* and Chapter 6 of this book for more information on creating customized library files.)
- WIDTH *n*** Sets terminal display width to *n* columns. For most terminal types, the default is 80 characters. On the PT200W terminal, the default is 132 characters.
- XOFF** This is the default. Causes EMACS to treat `Ctrl S` and `Ctrl Q` as PRIMOS stop and start print commands (which freeze and unfreeze the terminal display). When the `-XOFF` option is in effect, you may use the sequences `Ctrl X` `Ctrl Q`, `Esc` `S`, and `Ctrl X` `Ctrl S` to execute `quote`, `forward_search_command`, and `save_file`, respectively. Use the `-NOXOFF` option to restore the usual keybindings to `Ctrl S` and `Ctrl Q`.

## Abbreviating the Command Line

If you have an abbreviation file, you may define an abbreviation for your EMACS command line. To construct your abbreviation, use the following format:

```
ABBREV -ADD_COMMAND abbrev_name EMACS [%1%] -TTP terminal_type [options]
```

where *abbrev\_name* is the name of the abbreviation you will use, and *terminal\_type* indicates the type of terminal you are using. Other options may be selected from the preceding list.

See the *Prime User's Guide* for information about creating abbreviations.

## Saving Your Work

To save the changes you have made to one or more EMACS buffers, you must use one of the following commands:

▶ `Ctrl X Ctrl S` save\_file

`Ctrl X Ctrl S` automatically saves the text in your buffer in a file of the same name. It does not prompt you for a filename; the file must already exist.

▶ `Ctrl X Ctrl W` *pathname* mod\_write\_file

`Ctrl X Ctrl W` writes the text in your buffer to the file listed in the *pathname* that you specify. If you specify a file that does not exist, EMACS creates it for you. If you specify a file that already contains text, EMACS asks you if you want to overwrite the existing contents.

### Note

The prefix "mod" before an EMACS command name signifies a command that has been modified to make it more user-friendly. For example, the `write_file` command writes the contents of a buffer to the disk and overwrites an existing file with the same name without prompting you. `Mod_write_file` does not overwrite an existing file without first asking you if this is what you really want. Therefore, `mod_write_file` is bound to `Ctrl X Ctrl W` because it is more user-friendly than `write_file`. Generally, the "mod" version of a command is the one that is bound to a keypath.

▶ `Esc X` write\_file

This command is an extended command that works just like `mod_write_file`, described above, but it does *not* prompt you before overwriting existing contents of a file.

## Exiting From EMACS

You can exit from EMACS in the following ways:

- The `Ctrl X Ctrl C` command
- The `Ctrl P` character (PRIMOS break character)
- A recoverable error
- A forced logout condition

Each way is discussed below.

### The `Ctrl X Ctrl C` Command

You would normally exit from EMACS by issuing the `quit` command, or `Ctrl X Ctrl C`. This command gets you out of EMACS and back to the process from which you invoked EMACS, usually PRIMOS command level.

EMACS tries to prevent you from losing your work. If you have not saved modified buffers, or if you have changed your speed-type environment, EMACS prompts you and gives you a chance to save changes before you return to PRIMOS command level. You can save each modified buffer individually or use `Esc X save_all_files`. If you use the `save_all_files` command, those buffers that have no associated disk files are lost.

### The `Ctrl P` Character

You may leave EMACS by typing `Ctrl P` (the standard PRIMOS break character). See the break command entry in Chapter 3 for a complete description of the message that appears when you type `Ctrl P` and for instructions on reentering your current file without losing any editing changes.

### A Recoverable Error

An error message advises users to type either `REENTER` (or `REN`) or `START` (or `S`) to recover from a recoverable error that occurs within your EMACS environment.

### A Forced Logout Condition

If EMACS encounters a forced logout condition, it creates two types of recovery files in the user's initial attach point directory. The files are in the following formats:

1. `T$XXXXXXXXXXXX.EM.n`
2. `T$XXXXXXXXXXXX.EM.CPL`

For either type, the twelve *x*'s represent a unique, system-generated sequence of characters. The *n* suffix represents a positive integer.

EMACS creates one `.EM.n` file for every buffer containing unsaved changes. The file contains the contents of the buffer at the time of the forced logout.

EMACS writes a single `.EM.CPL` file per forced logout. This file, if executed, copies each `.EM.n` file into the corresponding disk file (providing that the buffer had an assigned default filename associated with it). The user receives messages that give information about what is occurring.

## A Locked Keyboard Condition

If, for some reason, your keyboard becomes locked during an EMACS session, first type `Ctrl Q` to undo a `Ctrl S` that you may have typed. If that does not work, try turning the terminal off for a few seconds, then on again. Use this technique when all else fails.

## Basic EMACS Commands

The following table contains ten EMACS commands that allow you to use EMACS at a rudimentary level. The next two chapters contain information about all of the current fundamental mode commands.

Table 1-1  
Ten Basic EMACS Commands

<i>Command</i>	<i>Description</i>
<code>Ctrl F</code>	Moves the cursor forward one character.
<code>Ctrl B</code>	Moves the cursor back one character.
<code>Ctrl N</code>	Moves the cursor to the next line.
<code>Ctrl Z</code>	Moves the cursor to the previous line.
<code>Ctrl D</code>	Deletes the character after point.
<code>Ctrl G</code>	Aborts the most recent command or exits the minibuffer.
<code>Ctrl P</code>	Interrupts an EMACS session; prompts you before breaking to PRIMOS command level.
<code>Ctrl K</code>	Kills the current line.
<code>Ctrl X</code> <code>Ctrl S</code>	Saves a file.
<code>Ctrl X</code> <code>Ctrl C</code>	Quits an EMACS session; prompts you if text changes are unsaved.

---

## 2

# Summary of EMACS Commands by Function

## Introduction

This chapter lists all the current EMACS commands, grouped by functional category. Within each category, commands are listed according to ease and frequency of use. Each group is introduced by an explanation of the circumstances in which its commands are used. Commands that fall into more than one category appear in more than one list, as appropriate.

The command list gives the name of the command; the keybinding, if any; and a brief description. If a command does not have a keybinding, you can execute it by typing `[Esc] [X]` first and responding to the prompt with the command name.

Appendix A contains a cross-reference list of EMACS commands arranged alphabetically by keybinding. For more information about Help commands, see Chapter 4, Online Help Facility. The speed-type facility is explained in Chapter 5, Speed-type. All of the commands in this chapter are discussed in Chapter 3, Dictionary of Commands.

The following categories are described in this chapter:

- Help commands
- Cursor movement commands
- Screen display commands
- Editing commands
- File management commands
- Buffer and window commands
- Macros
- PRIMOS command execution
- General modes
  - Dispatch tables
  - Overlay mode
  - Fill mode

- View mode
- Explore mode
- Cursor-function/number modes
- Environment commands
- Information commands
- Commands for slow terminals
- Library commands
- Speed-type commands
- Miscellaneous commands

## Help Commands

EMACS has several online help facilities designed to help you with any given operation during an EMACS session. Chapter 4, Online Help Facility, contains detailed information about these facilities. Use the `Ctrl_` (control underscore) command with its options to invoke online EMACS help during an editing session. The `Ctrl_` command is shown below.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>help_on_tap</code>	<code>Ctrl_</code>	Invokes help facility and lists command options.

Its options are as follows:

<i>Option</i>	<i>Command Name</i>	<i>Description</i>
A	<code>apropos</code>	Lists all commands related to an operation.
C	<code>explain_key</code>	Tells what a command keybinding does.
D	<code>describe</code>	Provides detailed information about commands and PEEL statements.
L		Displays last twenty characters you typed.
?		Lists help options.

You can also invoke some of the same help options by typing `[Esc] [X]`, followed by the command name, in answer to the prompt in the minibuffer. These help options are:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>apropos</code>		Alternate method of invoking the A help option
<code>describe</code>		Alternate method of invoking the D help option
<code>explain_key</code>	<code>[Esc] [?]</code>	Alternate method of invoking the C help option

## Cursor Movement Commands

You can move the cursor (and point) to a different place on the current screen, or you can move it forward or backward in the file enough so that the screen display changes to accommodate the move. You can also scroll the screen display vertically or horizontally without changing the cursor position.

The cursor indicates the location of point in the EMACS text. (Point lies between the cursor and the character that precedes the cursor.) The following commands move the cursor to different positions on your screen. These commands change the location of the cursor and point.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>forward_char</code>	<code>[Ctrl F]</code>	Moves cursor forward one character.
<code>back_char</code>	<code>[Ctrl B]</code>	Moves cursor back one character.
<code>next_line_command</code>	<code>[Ctrl N]</code>	Moves cursor down one line.
<code>prev_line_command</code>	<code>[Ctrl Z]</code>	Moves cursor up one line.
<code>begin_line</code>	<code>[Ctrl A]</code>	Moves cursor back to the beginning of a line.
<code>end_line</code>	<code>[Ctrl E]</code>	Moves cursor to the end of a line.
<code>repaint</code>	<code>[Ctrl X] [R]</code>	Moves cursor to first column of first line on screen.
<code>move_top</code>	<code>[Esc] [&lt;]</code>	Moves cursor to the beginning of the buffer.
<code>move_bottom</code>	<code>[Esc] [&gt;]</code>	Moves cursor to the end of the buffer.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>backward_sentence</code>	<code>Esc A</code>	Moves cursor to the beginning of the sentence.
<code>forward_sentence</code>	<code>Esc E</code>	Moves cursor to the end of the sentence.
<code>back_word</code>	<code>Esc B</code>	Moves cursor back to the beginning of a word.
<code>forward_word</code>	<code>Esc F</code>	Moves cursor forward a word.
<code>goto_line</code>	<code>Esc G</code>	Moves cursor to a line whose number is taken from the numeric argument.
<code>back_to_nonwhite</code>	<code>Esc M</code>	Moves cursor to first nonwhite character on line.
<code>backward_para</code>	<code>Ctrl X [</code>	Moves cursor to the beginning of the paragraph.
<code>forward_para</code>	<code>Ctrl X ]</code>	Moves cursor to the end of the paragraph.
<code>backward_clause</code>	<code>Ctrl X Ctrl Z Ctrl A</code>	Moves cursor back one clause.
<code>forward_clause</code>	<code>Ctrl X Ctrl Z Ctrl E</code>	Moves cursor forward one clause.

## Screen Display Commands

### Vertical Movement

EMACS can display only one screenful (21 lines) of a file at a time. When you move point vertically beyond the displayed section of your file, EMACS quickly generates a new display that is centered around the new position of point.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>next_page</code>	<code>Ctrl V</code>	Scrolls screen display forward 18 lines.
<code>back_page</code>	<code>Esc V</code>	Scrolls screen display back 18 lines.
<code>refresh</code>	<code>Ctrl L</code>	Refreshes the screen.

## Horizontal Movement

A typical terminal screen line can display 80 characters at a time. When you type a line that is longer than 80 characters, the cursor stops at the last column on your screen, but point continues to move as the text extends beyond the display. The column number of point appears in the first column of the status line.

To display extended text on the screen by wrapping it onto the next line, you can put fill mode in effect and then type `[Esc] [Q]`. However, if you are in wide-screen display mode and you want a line to extend and remain beyond 80 characters, you can use the following commands to view and edit the extended text.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
horiz_left	<code>Ctrl X</code> <code>[</code>	Shifts the current window to the left 40 spaces.
horiz_right	<code>Ctrl X</code> <code>]</code>	Shifts the current window to the right 40 spaces.
hcol		Sets or checks horizontal column.
set_hscroll		Prompts you for the hcol value used in horizontal scrolling.
hscroll		Uses current column position to set hcol.
reset		Resets windows and columns.

## Editing Commands

Editing commands enable you to insert, delete, move, search for, format, and save text. They also let you control your mode of text editing and your screen display. The commands in this section include the following categories:

- Inserting text
- Viewing nonalphabetic characters
- Deleting and restoring text
- The mark and the region
- The ring of marks
- Searching and replacing
- Case conversion
- Transposition
- Inserting new lines

- Tabs and indentation
- Formatting
- Saving text and exiting from EMACS
- Abort, break, and reexecute commands

## Inserting Text

This section describes commands and PI functions for typing text and controlling the screen display. In fundamental mode, the characters you type are inserted into the file to the left of the cursor position. Any other characters on the current line move to the right. To overwrite as you type, use overlay mode.

Fill mode (on or off) determines whether the screen display is governed by the right margin. When fill mode is on, as soon as you type a word that extends beyond the right margin, EMACS automatically inserts a carriage return and moves the word to the left margin of a new line.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>self_insert</code>	All the keys for all the printable characters.	Inserts a character into text buffer. (PI function)
<code>overlay_on</code>		Turns on overlay mode.
<code>overlay_off</code>		Turns off overlay mode.
<code>fill_on</code>		Turns on fill mode.
<code>fill_off</code>		Turns off fill mode.
<code>set_right_margin</code>	<code>Ctrl X F</code>	Sets right margin to specified value.
<code>set_left_margin</code>		Sets left margin to position of cursor.
<code>take_left_margin</code>	<code>Ctrl X .</code>	Sets left fill margin to column containing the cursor.
<code>tell_right_margin</code>		Gives current setting of right margin.
<code>tell_left_margin</code>		Gives current setting of left margin.
<code>fill_para</code>	<code>Esc Q</code>	Fills paragraph according to set margins in fill mode.
<code>open_line</code>	<code>Ctrl O</code>	Inserts a carriage return after the cursor without moving the cursor.

## Viewing Nonalphabetic Characters

To insert a nonalphabetic character, such as `Ctrl L`, `Esc`, `TAB`, or `DEL`, you must precede it with the `Ctrl Q` command. This command tells EMACS to quote the character following it instead of interpreting that character as an EMACS command. These special characters constitute a "blotch" character which your terminal interprets as a question mark or a rectangular block.

On some terminals, `Ctrl Q` acts as the standard PRIMOS "start print" command instead of an EMACS command. If this is the case on your terminal, type `Ctrl X Q` instead. Neither `^q_quote_command` nor `quote_command` can be executed as an extended command.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>^q_quote_command</code>	<code>Ctrl Q</code>	Inserts a nonalphabetic character into your text.
<code>quote_command</code>	<code>Ctrl X Q</code>	Identical to <code>^q_quote_command</code> .

## Deleting and Restoring Text

The commands in this section perform basic deletion tasks. The deleted text is not saved.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>delete_char</code>	<code>Ctrl D</code>	Deletes character after point.
<code>rubout_char</code>	<code>Ctrl H</code>	Deletes preceding character.
<code>delete_blank_lines</code>	<code>Ctrl X Ctrl O</code>	Deletes blank lines immediately above and below cursor.
<code>delete_region</code>		Deletes current marked region; does not save in kill ring. (See next section.)
<code>delete_buffer</code>		Deletes current buffer; does not save in kill ring.
<code>merge_lines</code>	<code>Esc ^</code>	Merges two lines together.
<code>white_delete</code>	<code>Esc \</code>	Deletes all whitespaces around cursor.
<code>leave_one_white</code>	<code>Esc SPACE</code>	Deletes all but one whitespace before cursor.

For most commands that can delete more than one character, EMACS saves the deleted text by placing it on a stack called the *kill ring*. The ring contains the last 11 blocks of text that you killed. If you kill two blocks of text in a row without using any commands in between, they are saved as one block on the kill ring. You can "yank" text off the kill ring and restore it to any position in any buffer at any time. You can also look at the text on the kill ring and delete a text block that you do not want. There is only one kill ring, and it is not affected when you switch buffers.

The following commands let you delete or copy text and save it in the kill ring. The following command lets you view the contents of the kill buffers:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<b>delete_word</b>	<b>Esc D</b>	Deletes one word at a time.
<b>rubout_word</b>	<b>Esc Ctrl H</b>	Deletes previous word.
<b>kill_line</b>	<b>Ctrl K</b>	Deletes text from cursor to end of current line.
<b>backward_kill_line</b>	<b>Ctrl X Ctrl K</b>	Deletes text from cursor to beginning of line.
<b>forward_kill_sentence</b>	<b>Esc K</b>	Deletes sentence from cursor to end.
<b>backward_kill_sentence</b>	<b>Ctrl X Ctrl H</b>	Deletes sentence from cursor to beginning.
<b>forward_kill_clause</b>	<b>Ctrl X Ctrl Z Ctrl K</b>	Deletes forward one clause.
<b>backward_kill_clause</b>	<b>Ctrl X Ctrl Z Ctrl H</b>	Deletes backward one clause.
<b>kill_rest_of_buffer</b>	<b>Esc Ctrl D</b>	Deletes all text from cursor to end of buffer.
<b>kill_region</b>	<b>Ctrl W</b>	Moves region between mark (see next section) and cursor to kill ring.
<b>copy_region</b>	<b>Esc W</b>	Copies region onto the kill ring.

The following command lets you view the contents of the kill buffers:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<b>view_kill_ring</b>	<b>Ctrl X Ctrl Z K</b>	Lets you view contents of kill buffers.

The following commands let you restore text from the kill ring:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<b>yank_region</b>	<b>Ctrl Y</b>	Restores the last text deleted.
<b>yank_replace</b>	<b>Esc Y</b>	Replaces text restored by <b>yank_region</b> with previously deleted text.
<b>yank_kill_text</b>	<b>Ctrl X Ctrl Z Ctrl Y</b>	Inserts at point text saved by <b>view_kill_ring</b> .

## The Mark and the Region

Some EMACS commands operate within a defined portion of the buffer called a *region*. The region contains the text between the *mark* and the current position of point. To define a region, you first set a mark, which is invisible on the screen, and then move the cursor to the place where you want the region to end. Use `exchange_mark` to confirm the boundaries of the region. This command does not alter the region. Subsequently, any command that you issue that operates on regions of text will take effect between the mark and the point only. The following commands control the mark:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>mark</code>	<code>Ctrl @</code>	Places mark at cursor.
	<code>Ctrl U</code> <code>Ctrl @</code>	Moves cursor back one position of mark on ring of marks. (See <code>mark</code> command definition in Chapter 3.)
<code>setmark</code>		Sets mark at current cursor position.
<code>exchange_mark</code>	<code>Ctrl X</code> <code>Ctrl X</code>	Exchanges cursor with mark.
<code>mark_end_of_word</code>	<code>Esc</code> <code>@</code>	Places mark at the end of the current word.
<code>mark_top</code>	<code>Ctrl X</code> <code>Ctrl Z</code> <code>&lt;</code>	Places mark at top of buffer.
<code>mark_bottom</code>	<code>Ctrl X</code> <code>Ctrl Z</code> <code>&gt;</code>	Places mark at bottom of buffer.
<code>mark_para</code>	<code>Esc</code> <code>H</code>	Marks paragraph.
<code>mark_whole</code>	<code>Ctrl X</code> <code>H</code>	Marks whole buffer as region.

The following commands manipulate regions:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>append_to_file</code>	<code>Ctrl X</code> <code>Ctrl Z</code> <code>A</code>	Appends region to file.
<code>prepend_to_file</code>	<code>Ctrl X</code> <code>Ctrl Z</code> <code>P</code>	Prepends region to file.
<code>append_to_buf</code>	<code>Ctrl X</code> <code>A</code>	Appends region to buffer.
<code>kill_region</code>	<code>Ctrl W</code>	Moves region between mark and cursor to kill ring.
<code>prepend_to_buf</code>	<code>Ctrl X</code> <code>P</code>	Prepends region to buffer.
<code>copy_region</code>	<code>Esc</code> <code>W</code>	Copies region onto the kill ring.

## The Ring of Marks

As well as delimiting a region, the mark is also useful for remembering a position in your text. EMACS stores 10 previous locations of the mark on a ring. The `Ctrl U Ctrl @` command returns the cursor to the location of the preceding mark. You can keep cycling through the ring and back to the most recent mark location.

The following commands work with the ring of marks.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
	<code>Ctrl U Ctrl @</code>	Moves cursor back one position of mark on ring of marks.
<code>pushmark</code>		Sets new mark and pushes the old one onto the mark stack.
<code>popmark</code>		Pops a mark off the top of the mark stack.

## Searching and Replacing

The following EMACS commands are used for searching for a character string with the option of replacing it with another. When EMACS performs any search for a character string, case matching occurs by default. This means that if you ask EMACS to locate a character string that you have typed in lowercase, it ignores any occurrence of the string that contains uppercase letters. If you do not want case matching to occur during replace operations, you can turn it off with the `case_replace_off` command (listed below) or the `case_off` command.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>^s_forward_search_command</code>	<code>Ctrl S</code>	Searches forward for specified string.
<code>forward_search_command</code>	<code>Esc S</code>	Identical to <code>Ctrl S</code> command.
<code>forward_search_again</code>		Searches forward again.
<code>reverse_search_command</code>	<code>Ctrl R</code>	Searches backward for specified string.
<code>reverse_search_command</code>	<code>Esc R</code>	Identical to <code>Ctrl R</code> command.
<code>reverse_search_again</code>		Searches backward again.
<code>case_off</code>		Causes EMACS not to distinguish between cases of letters during searching.
<code>case_on</code>		Causes EMACS to distinguish between cases of letters during searching.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
case?		Tells you whether cases are distinguished during searching.
replace		Searches for a string and replaces all instances with another.
query_replace	<span>Esc</span> <span>%</span>	Replaces the occurrence of one string with another.
case_replace_off		Treats uppercase and lowercase letters the same during replace operations.
case_replace_on		Distinguishes uppercase letters from lowercase during replace operations.
case_replace?		Tells whether cases are distinguished during replace operations.

## Case Conversion

The commands in this section perform case conversion for words and regions of text.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
lowercase_word	<span>Esc</span> <span>L</span>	Changes word into lowercase.
uppercase_word	<span>Esc</span> <span>U</span>	Changes word into uppercase.
capinitial	<span>Esc</span> <span>C</span>	Capitalizes first letter of word.
lowercase_region	<span>Ctrl X</span> <span>Ctrl L</span>	Converts region to lowercase.
uppercase_region	<span>Ctrl X</span> <span>Ctrl U</span>	Converts region to uppercase.

## Transposition

The commands in this section transpose characters or words.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
twiddle	<span>Ctrl T</span>	Inverts position of two characters preceding cursor.
transpose_word	<span>Esc</span> <span>T</span>	Inverts position of words before and after cursor.

## Inserting New Lines

You may add new lines to your file, before or after a specific line, by using either of the commands in this section. EMACS separates lines by inserting the carriage return (cr) character at the end of a line to act as a line separator. You can insert and delete the cr character as you do any other EMACS character, even though you cannot see it on the screen. When EMACS creates a new line, the cr is the only character that line contains. As more characters are added to the line, they are inserted before the cr character.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
cr	<input type="button" value="Return"/>	Inserts carriage return into the buffer at point.
open_line	<input type="button" value="Ctrl O"/>	Inserts a carriage return at cursor without moving the cursor.

## Tabs and Indentation

This section describes EMACS commands for indenting text and using tabs.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
cret_indent_relative	<input type="button" value="Ctrl X"/> <input type="button" value="Return"/>	Inserts carriage return at current cursor position; moves cursor to new line, indenting left margin to same column as previous line.
indent_relative	<input type="button" value="Esc"/> <input type="button" value="I"/>	Indents current line with one directly above it.
indent_to_fill_prefix	<input type="button" value="Esc"/> <input type="button" value="Ctrl I"/>	Moves line to left margin.
split_line	<input type="button" value="Esc"/> <input type="button" value="Ctrl O"/>	Breaks line at cursor and indents next line at same column.
center_line	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl Z"/> <input type="button" value="S"/>	Centers current line of text.
type_tab	<input type="button" value="Ctrl I"/>	Moves cursor forward to next tab stop.
insert_tab	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl I"/>	Inserts spaces to next tab stop.
back_tab		Moves cursor back a tab stop.
settab		Sets tab to any value.
set_tab		Same as settab.
set_tabs		Same as settab.
tablist		Sets tab positions to a series of set numbers.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
settabs_from_table or setft		Sets tabs based on column position of words.
default_tabs		Restores tab position to every five spaces.
global_tabs		Activates global (default) tabs.
local_tabs		Activates local (to buffer) tabs.
save_tab		Saves current tab positions in a file.
get_tab		Retrieves previously saved tabs.
which_tabs		Tells which tabs are in use.

## Formatting

The following commands set margins and justify text within fill mode. (Fill mode is explained in the General Modes section, later in this chapter.)

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
set_right_margin	<span>Ctrl X</span> <span>F</span>	Sets the right margin to the column you specify.
set_left_margin		Sets left margin to position of the cursor.
take_right_margin	<span>Ctrl X</span> <span>Ctrl Z</span> <span>F</span>	Takes right margin from current cursor position. Column 10 is cutoff point.
take_left_margin	<span>Ctrl X</span> <span>.</span>	Sets left fill margin to column containing the cursor. Useful when doing a <b>fill_para</b> .
tell_right_margin		Gives current setting of right margin.
tell_left_margin		Gives current setting of left margin.
fill_para	<span>Esc</span> <span>Q</span>	Fills paragraph according to set margins.
untidy		Unjustifies a paragraph.

## Saving Text and Exiting From EMACS

You probably want to save new text or changes made to an existing file before you exit from EMACS. If you quit without saving, EMACS notifies you which buffers have been modified and prompts you to verify that you want to exit without saving.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
save_file	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl S"/> or <input type="button" value="Ctrl X"/> <input type="button" value="S"/>	Saves file in PRIMOS under its current name.
save_all_files		Saves all files you modify.
mod_write_file	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl W"/>	Prompts you before writing buffer to file.
write_file		Writes buffer to named file. Does not prompt.
quit	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl C"/>	Returns you to PRIMOS.

## Abort, Break, and Reexecute Commands

You can abort an EMACS command if you have not finished typing it or have not typed . You can break from the EMACS session and select one of three actions offered by the prompt. You may also reexecute the last command that you typed.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
abort_command	<input type="button" value="Ctrl G"/>	Aborts a command or exits minibuffer.
ignore_prefix	<input type="button" value="Ctrl X"/> <input type="button" value="Ctrl G"/> or <input type="button" value="Ctrl X"/> <input type="button" value="Ctrl Z"/> <input type="button" value="Ctrl G"/>	Aborts a command.
PRIMOS break (Not an EMACS command.)	<input type="button" value="Ctrl P"/>	Breaks from EMACS session and prompts for choice from three options: Q - quit (return to PRIMOS command level), A - abort current command and continue EMACS session, C - continue EMACS session.
reexecute	<input type="button" value="Ctrl C"/>	Reexecutes last command.

## File Management Commands

EMACS is designed to edit text files (files of characters) consisting of lines terminated by a new line character.

If your user process has not been allocated an adequate number of dynamic segments, you will be plagued by EMACS telling you **NOT ENOUGH SEGMENTS** and having many of your commands (e.g., `find_file`, `insert_buffer`) aborted. To avoid this problem, ask your System Administrator to set your allowed number of dynamic segments to at least 100.

### Note

EMACS uses segments in the 2000 range for its libraries. At Rev. 19.4 and later revisions, it allocates storage in the dynamic segments.

In EMACS, you can work with several different versions of the same file, or with several different files, in one editing session. EMACS places the different versions or files in *buffers*, or storage areas.

When you create an EMACS file, the file exists only in a temporary EMACS storage area, called a buffer, until you save or write the file. If you exit from EMACS without saving or writing, the new file is lost. Similarly, when you edit a file, you are not editing the actual disk file. You are editing a copy of the file that EMACS has placed in a buffer. To make permanent changes to the disk file, you must save the changes that you have made. If you edit a file and exit from EMACS without saving the changes, the disk file remains as it was before you began the current editing session.

The commands for reading and writing files are listed below.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>find_file</code>	<code>Ctrl X</code> <code>Ctrl F</code>	Finds a file and reads it into a buffer named after the filename found.
<code>read_file</code>	<code>Ctrl X</code> <code>Ctrl R</code>	Reads a file into current buffer.
<code>save_file</code>	<code>Ctrl X</code> <code>Ctrl S</code> or <code>Ctrl X</code> <code>S</code>	Saves file in PRIMOS under its current name.
<code>write_file</code>		Writes buffer to named file. Does not prompt.
<code>mod_write_file</code>	<code>Ctrl X</code> <code>Ctrl W</code>	Prompts you before overwriting file with buffer.
<code>save_all_files</code>		Saves all files you modify.
<code>unmodify</code>	<code>Esc</code> <code>~</code>	Treats buffer as if it were not modified.
<code>insert_file</code>	<code>Ctrl X</code> <code>I</code>	Inserts file at cursor.

## Buffer and Window Commands

### Buffers

EMACS uses buffers as workspaces to hold and organize your files during an editing session. The number of buffers that you may use in an EMACS editing session is determined by the system resources.

Each buffer in EMACS has a name, which is always displayed in parentheses on the status line. Normally, a buffer has the same name as the file it contains, but you can also create and name a buffer independently. (A buffer name cannot be null.)

Even though you move from buffer to buffer, EMACS maintains the state of the text in each buffer: which modes are in effect, the current cursor position, and whether there are unsaved changes to the text.

The following commands let you manipulate buffers.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>mod_select_buf</code>	<code>Ctrl X B</code>	Moves you to the specified buffer.
	<code>Ctrl X B Return</code>	Returns you to the previous buffer.
<code>select_buf</code>		Moves you to the specified buffer. (Same as <code>mod_select_buf</code> .)
<code>list_buffers</code>	<code>Ctrl X Ctrl B</code>	Lists all active buffers.
<code>next_buf</code>	<code>Esc N</code>	Replaces current buffer with the next buffer. (See EMACS Command Conventions section in Chapter 1 for restrictions on using <code>Esc N</code> in the EMACS SUIX.)
<code>prev_buf</code>	<code>Esc P</code>	Replaces current buffer with the previous buffer.
<code>insert_buf</code>	<code>Ctrl X Ctrl Z I</code>	Inserts specified buffer at cursor.
<code>get_bufname</code>		Inserts current buffer name in buffer at cursor.

### Windows

EMACS allows you to divide your screen into horizontal or vertical sections, called *windows*, so that you may compare files or edit more than one file at a time. The *EMACS Primer* contains detailed instructions for working with windows in an editing session. Also, for more information

about the commands in this section, see Chapter 3, Dictionary of EMACS Commands. This section describes the commands for using multiple windows.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>mod_split_window</code>	<code>Ctrl X</code> <code>2</code>	Divides current window horizontally into two windows.
<code>split_window</code>		Divides current window horizontally into two windows. (Same as <code>mod_split_window</code> .)
<code>mod_split_window_stay</code>	<code>Ctrl X</code> <code>3</code>	Splits current window; keeps cursor in first window.
<code>split_window_stay</code>		Splits current window; keeps cursor in first window. (Same as <code>mod_split_window_stay</code> .)
<code>other_window</code>	<code>Ctrl X</code> <code>O</code>	Moves cursor between current window and previous window.
<code>select_any_window</code>	<code>Ctrl X</code> <code>4</code>	Cycles cursor through all windows.
<code>mod_one_window</code>	<code>Ctrl X</code> <code>1</code>	Transforms a split screen into one.
<code>one_window</code>		Transforms a split screen into one. (Same as <code>mod_one_window</code> .)
<code>scroll_other_forward</code>	<code>Esc</code> <code>Ctrl V</code>	Scrolls other window forward.
<code>scroll_other_backward</code>	<code>Ctrl X</code> <code>V</code>	Scrolls other window backward.
<code>vsplit</code>		Splits current window vertically at point into two windows.

## Macros

A macro is a group of EMACS commands that is constructed and saved so that it can be executed as a single command. By using a macro, you can define a repetitive task as one command to avoid typing the same sequence of commands over and over again. For example, if you find that you need to type `Ctrl N` `Ctrl D` forty times, you can define a keyboard macro that executes `Ctrl N` `Ctrl D` and then give it an argument to repeat the command thirty-nine more times. The following example shows the keystrokes you would use to define this macro, with explanations of each group of keystrokes.

<i>Keystrokes</i>	<i>Description</i>
<code>Ctrl X</code> <code>(</code>	Starts collecting the macro.
<code>Ctrl N</code> <code>Ctrl D</code>	The actual macro.
<code>Ctrl X</code> <code>)</code>	Stops collecting the macro.

Now, to execute the macro thirty-nine times, type:

`Esc 3 9 Ctrl X E`

Keyboard macros are useful for accomplishing specific tasks in EMACS. Chapter 12 in the *EMACS Primer* gives detailed instructions for defining and executing EMACS keyboard macros. However, if you want to write a function to do things that cannot be done using ordinary EMACS commands, you must use PEEL, the Prime EMACS Extension Language. Chapter 2 of the *EMACS Extension Writing Guide* explains how to construct a macro and convert it to PEEL source code.

#### Note

An `expand_macro` command may not duplicate the actual sequence of keystrokes that you entered from the keyboard if the sequence includes bound function keys. For example, if a user creates a keyboard macro using the Global Replace key, EMACS is inconsistent in retaining the search and replace strings within the macro.

The following commands are used for defining, executing, and saving keyboard macros.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>collect_macro</code>	<code>Ctrl X (</code>	Starts collecting macro.
<code>finish_macro</code>	<code>Ctrl X )</code>	Stops collecting macro.
<code>execute_macro</code>	<code>Ctrl X E</code>	Executes current macro.
<code>expand_macro</code>		Expands a stored macro into PEEL source code with a given macro name.
<code>set_permanent_key</code>		Binds function to a key for an entire session.
<code>set_key</code>		Binds function to a key on a per-buffer basis.
<code>pl</code>		Compiles and executes source code in current buffer.
<code>pl_minibuffer</code>	<code>Esc Esc</code>	Invokes PEEL.

## PRIMOS Command Execution

EMACS lets you execute a PRIMOS command while remaining in the EMACS environment. To do this, you use the `primos_command` command (`Ctrl X Ctrl E`) and any of its prefixes. The `Ctrl X Ctrl E` command is shown below:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>primos_command</code>	<code>Ctrl X Ctrl E</code>	Executes a PRIMOS command.

Its options are as follows:

<i>Option Name</i>	<i>Form</i>	<i>Description</i>
<code>primos_external</code>	no prefix	Command is executed as a phantom.
<code>primos_internal_como</code>	!	Output is displayed in <code>file_output</code> buffer.
<code>primos_internal_screen</code>	!!	Output of internal command is displayed on new screen.

When you type `Ctrl X Ctrl E` or `Esc X primos_command`, EMACS displays the PRIMOS command prompt. When you select either ! or !! in response to the prompt, the option must precede the selected PRIMOS command.

## General Modes

When you first initialize EMACS, you are in fundamental mode by default unless you specify the EMACS SUI or the EMACS SUIX on your command line. In fundamental mode, all commands are bound to certain characters on your keyboard. You execute EMACS commands by typing these characters. However, depending on your needs, you can redefine fundamental mode keybindings to change the way certain commands work. This feature in EMACS is called changing modes.

You can change modes by asking for a different mode from your current EMACS file, or you can place mode commands in your startup library file so that they are turned on when you invoke EMACS. The section on file hooks in Chapter 6 tells you how to set your user type so that when you are working on a specific kind of document, the mode that you need to edit that document is turned on automatically as you enter EMACS.

An example of a mode that you can request is fill mode. This mode rebinds the space character to a command that checks to see if you have typed past the right margin. If you have, it breaks the line at the right margin and moves the last word you typed to the first column of a new line.

Modes take effect only in the buffers you specify. For example, you could use fill mode in one buffer, switch to a new buffer and use overlay mode, and then switch to a third buffer and use

view mode. Or, you could use more than one mode in one buffer. EMACS "remembers" which modes are in effect in which buffer so you can switch back and forth without having to retype the mode commands.

Whenever you put a mode in effect, its name appears in the status line to remind you that certain commands behave differently while you are using that mode.

## Dispatch Tables

In EMACS, functions are bound to keystrokes through a dispatch table. There is a special dispatch table for each mode. When a mode is in effect, the dispatch table for that mode is used first. When that mode is no longer in effect, the dispatch table definitions for the latest mode are read. For more information about dispatch tables, see the *EMACS Extension Writing Guide*.

The following four general commands are helpful when using modes:

<i>Command Name</i>	<i>Description</i>
<code>set_mode</code>	Sets buffer to specified mode.
<code>set_mode_key</code>	Binds function to a key on a per-mode basis.
<code>tell_modes</code>	Displays all modes for buffer.
<code>all_modes_off</code>	Turns all modes off.
<code>fundamental</code>	Turns on fundamental mode.

## Overlay Mode

In overlay mode, the characters you type overwrite existing text.

### Note

When you are in overlay mode and two-dimensional mode, each line is treated as an independent entity; you cannot use `Ctrl D` (`delete_char`) to delete a carriage return character at the end of the current line, or `Ctrl H` (`rubout_char`) to delete a carriage return character at the end of the previous line. However, you can use `Ctrl K` (`kill_line`) to delete a carriage return character at the end of the current line.

The commands and functions for turning overlay mode on and off are listed below.

<i>Command Name</i>	<i>Description</i>
<code>overlay_on</code>	Turns overlay mode on.
<code>overlay_off</code>	Turns overlay mode off.
<code>overlay_rubout</code>	Erases previous character in overlay mode.
<code>overlayer</code>	Overlays typed character. (PI function)

## Fill Mode

In fill mode, as soon as you type a word that extends beyond the right margin, EMACS automatically inserts a carriage return and moves the word to the left margin of a new line. The following commands turn fill mode on and off.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>fill_on</code>		Turns on fill mode.
<code>fill_off</code>		Turns off fill mode.
<code>wrap</code>	<code>SPACE</code> or <code>Return</code>	Inserts carriage return.

## View Mode

When view mode is in effect, the current buffer becomes a read only buffer. If you try to use any commands that modify text, EMACS responds with an error message. View mode also has some options to make it easier to move through the buffer.

The commands for turning view mode on and off are listed below.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>view_on</code>		Turns view mode on.
<code>view_off</code>	<code>Ctrl U</code> <code>Ctrl X</code> <code>Ctrl V</code>	Turns view mode off.
<code>view_file</code>	<code>Ctrl X</code> <code>Ctrl V</code>	Allows viewing of file. Only view mode command options can be used.
<code>view</code>		Same as <code>view_file</code> except that bound function keys can be used.

The following view mode command options (keystrokes) allow you to move through the buffer in view mode:

<i>Option</i>	<i>Description</i>
<code>SPACE</code>	Moves cursor forward a screen.
<code>B</code>	Moves cursor back one screen.
<code>&gt;</code>	Moves cursor to start of text.
<code>&lt;</code>	Moves cursor to end of text.
<code>R</code>	Executes reverse search.
<code>S</code>	Executes forward search.

## Explore Mode

Explore mode allows you to look through and make changes to directories and subdirectories without leaving EMACS. The following command activates explore mode:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<b>explore</b>	<b>Ctrl X</b> <b>D</b>	Turns on explore mode.

You can use the standard cursor movement commands when you are looking at a directory listing in the explore buffer. In addition, there are options having special meaning in explore mode. You can execute these keystroke options only from the explore buffer.

<i>Option</i>	<i>Description</i>
<b>H</b> or <b>?</b>	Displays a list of explore mode commands.
<b>@</b>	Attaches to current explore directory.
<b>A</b>	Displays attributes on screen.
<b>C</b> or <b>N</b>	Creates a file.
<b>D</b> or <b>G</b>	Displays directory or file specified by cursor.
<b>K</b>	Deletes file specified by cursor.
<b>L</b>	Updates information for current directory.
<b>O</b>	Sets spool options for explore mode.
<b>P</b>	Dives into a passworded directory.
<b>Q</b>	Quits and returns you to previous buffer.
<b>R</b>	Renames or relocates file/directory.
<b>S</b>	Spools file specified by cursor.
<b>U</b>	Moves you up a level in directory.
<b>Ctrl X</b> <b>U</b>	Moves you to the next highest directory level from within a file.

When you are looking at actual text in a file, any of the normal EMACS commands work.

You can leave explore mode by issuing one of the buffer or file commands, for example, **Esc** **P** (**prev\_buf**), or by displaying a file. For more information about explore mode, see the Dictionary of EMACS Commands section in Chapter 3.

## Cursor-Function/Number Modes

The group of keys on the lower right side of the PT200 terminal's keyboard is called the Cursor-Function/Number (CF/N) pad. When you are in the EMACS SUIX, you can change these keys from cursor-function mode to number mode with the Num Lock key, located at the far left of the top row of the pad.

When you are in cursor-function mode, the pad sends several different Esc sequences to the host, which the EMACS SUI uses for cursor control and similar functions. You can bind your own functions to the CF/N pad while in cursor-function mode. Number mode (which causes the Num Lock light to turn on) lets you use keys on the pad to type numbers and other symbols etched on the sides of the keys.

On the PT200, when you enter EMACS in fundamental mode, the value of your CF/N pad switch determines whether the pad is in cursor-function or number mode.

## Environment Commands

This section lists the EMACS commands that create environments. The list includes commands for these environments:

- Line numbering
- Continuous lines environment (two-dimensional mode)

### Line Numbering

EMACS does not use line numbering by default. You can enable and disable line numbering by using the following commands:

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
#on		Numbers lines on screen.
#off		Takes line numbers off screen.
#		Tells you if line numbering is in effect.
goto_line	<span>Esc</span> <span>G</span>	Moves cursor to a specified line.

### Continuous Lines Environment (Two-dimensional Mode)

This environment is put into effect automatically as part of overlay mode. When you use Ctrl N (`next_line_command`) or Ctrl Z (`prev_line_command`) in the continuous lines environment, the cursor maintains its column position, regardless of the length of the new line. To invoke this environment, use the following commands.

<i>Command Name</i>	<i>Description</i>
2don	Turns two-dimensional mode on.
2doff	Turns two-dimensional mode off.
2d	Tells you if two-dimensional mode is on.

## Information Commands

Some EMACS commands give you information about what is going on in EMACS and others give you general information such as the date and time. All commands in this section are discussed in detail in Chapter 3.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>display_debug</code>		Displays information about debug. A single keystroke clears the display.
<code>display_buffer</code>		Displays information about current buffer.
<code>display_terminal</code>		Displays information about your terminal.
<code>display_window</code>		Displays information about current window.
<code>tell_position</code>	<code>Ctrl X =</code>	Minibuffer message gives line and cursor position.
<code>tell_modes</code>		Displays all modes for buffer.
<code>get_bufname</code>		Inserts current buffer name into buffer at cursor.
<code>get_filename</code>	<code>Ctrl X Ctrl Z Ctrl F</code>	Inserts pathname of file in current buffer into buffer at cursor.
<code>dt</code>		Inserts date and time: <b>12/29/86 09:59:37</b>
<code>date</code>		Inserts day, month, year: <b>MON, 29 DEC 1986</b>
<code>europe_dt</code>		Inserts date in the European format: <b>29/12/86</b>
<code>trim_dt</code>		Inserts month, day, year: <b>12/29/86</b>
<code>trim_date</code>		Inserts day, month, year: <b>29 Dec 1986</b>
<code>sort_dt</code>		Inserts year, day, month: <b>86/12/29</b>
<code>insert_version</code>		Inserts version of EMACS used.
<code>new_features</code>		Displays the Info file found in EMACS*>INFO>NEW_FEATURES_INFO.
<code>wallpaper</code>		Lists all EMACS commands and functions.

## Commands for Slow Terminals

EMACS is designed to be used on a display terminal on a line that supports at least 1200 baud. If you are using EMACS at a slower baud rate, you will find it helpful to use the commands described in this section.

Because EMACS is a full screen editor, nearly every command you type causes the terminal display to change. This information exchange between the terminal and the computer can be very time consuming at a slow baud rate.

The commands described below are designed to enhance the performance of EMACS on a slow terminal.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>toggle_redisplay</code>	<code>Ctrl X</code> <code>Ctrl T</code>	Toggles the redisplay mode.
<code>view_lines</code>	<code>Ctrl X</code> <code>Ctrl Z</code> <code>Ctrl V</code>	Views lines toggled off.

## Library Commands

The following commands are used when compiling and executing library files. See Chapter 6 for information about creating, compiling, loading, and executing library files.

<i>Command Name</i>	<i>Description</i>
<code>dump_file</code>	Partially compiles a PEEL file in the current buffer to a fasdump file with .EFASL suffix.
<code>load_compiled</code>	Loads a fasdump file saved with the <code>dump_file</code> command.
<code>load_pl_source</code>	Compiles and loads the source code in a PL source file.
<code>set_user_type</code>	Sets the user type for file hook procedure.

## Speed-type Commands

The speed-type facility offers the potential for substantial savings in time and effort during text entry. Refer to Chapter 5, Speed-type, for a complete explanation of how to use this EMACS facility.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>spd_on</code>		Turns on speed-type abbreviations.
<code>spd_off</code>		Turns off speed-type abbreviations.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>spd_add</code>		Adds a <code>speed_type</code> abbreviation.
<code>spd_list</code>		Gives information about specific symbol.
<code>spd_save_file</code>		Saves changes made to current speed-type environment.
<code>spd_delete</code>		Deletes speed-type abbreviation.
<code>spd_list_file</code>		Lists abbreviations for specific file.
<code>spd_list_all</code>		Gives information about all speed-type abbreviations.
<code>spd_load_file</code>		Loads a speed-type abbreviation file.
<code>spd_compile</code>		Compiles current buffer into speed-type file.
<code>spd_add_modal</code>		Adds abbreviation for specific mode.
<code>spd_add_region</code>	<code>Ctrl X +</code>	Defines a region as expansion of speed-type abbreviation.
<code>spd_unexpand</code>	<code>Ctrl X -</code>	Removes expansion from current buffer.
<code>back_place_holder</code>	<code>Ctrl X &lt;</code>	Moves to previous speed-type place holder.
<code>forward_place_holder</code>	<code>Ctrl X &gt;</code>	Moves to next speed-type place holder.

## Miscellaneous Commands

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>extend_command</code>	<code>Esc X</code>	Prefix that lets you execute any EMACS command.
<code>multiplier</code>	<code>Ctrl U</code>	Multiplies prefix count by 4.
<code>abort_minibuffer</code>	<code>Ctrl G</code>	Aborts minibuffer. Used in minibuffer mode.

<i>Command Name</i>	<i>Keybinding</i>	<i>Description</i>
<code>exit_minibuffer</code>	<code>Return</code>	Exits minibuffer. Used in minibuffer mode.
<code>yank_minibuffer</code>	<code>Esc</code> <code>Ctrl Y</code>	Inserts the response to the previous minibuffer prompt into your current buffer.

---

# 3

## Dictionary of EMACS Commands

### Introduction

This chapter is a dictionary. It contains all of the standard EMACS commands, arranged alphabetically by command name. Names that begin with nonalphabetic characters are at the beginning of the list. If the command is normally bound to a keypath, the keybinding is displayed on the same line as the command name. The keybindings in this chapter are for EMACS fundamental mode.

#### Note

When you type a command name or a keybinding, you can use either uppercase or lowercase letters.

### Dictionary of Commands

#### ▶ #

This command tells whether the line numbering environment is in effect. One of the following messages appears in the minibuffer:

```
Line numbers are off
```

or

```
Line numbers are on
```

#### ▶ #off

This command turns off line numbering and removes the number from the left of each line on your screen.

▶ **#on**

This command turns on line numbering, so that the number of each line appears at the left of your screen. These numbers are not actually part of your file.

▶ **2d**

This command tells whether the continuous lines environment (two-dimensional mode, described in Chapter 2) is on. One of the following messages appears in the minibuffer:

2d is on

or

2d is off

▶ **2doff**

This command turns the continuous lines environment off.

▶ **2don**

This command turns the continuous lines environment on. If you move the cursor to a line that is shorter than the current one, this environment keeps the cursor in the column it just left.

This environment is automatically put in effect as part of overlay mode.

**Note**

When you are in overlay mode, with 2d in effect, each line is treated as an independent entity, so that you cannot use **Ctrl D** to delete a carriage return character at the end of the current line, nor can you use **Ctrl H** to delete a carriage return character at the end of the previous line. However, you can use **Ctrl K** to delete a carriage return character at the end of the current line.

▶ **^q\_quote\_command**

**Ctrl Q**

If you need to insert a nonprinting character that you type literally, such as **Ctrl F**, **Esc**, **TAB**, or **DEL**, you must precede it with the **Ctrl Q** command. This command tells EMACS to "quote" the special character following it instead of interpreting that character as an EMACS command. These special characters are usually displayed on your screen as question marks (?) or as rectangular blocks.

**Note**

If you do not use `-NOXOFF` as an option on your EMACS initialization line, `Ctrl Q` acts as the standard PRIMOS start print command instead of an EMACS command. If this is the case, type `Ctrl X Q` to effect the quote.

Neither `^q_quote_command` nor `quote_command` can be executed as an extended command.

**► `^s_forward_search_command`**`Ctrl S`

This command is the same as the `forward_search_command` command. It searches forward in your current buffer for the first occurrence of the string of characters specified in response to the minibuffer prompt. If the cursor is in the middle of a file, EMACS searches only the text following the cursor.

When you type `Ctrl S`, you see the following prompt in the minibuffer area:

Forward Search:

Enter the string of characters you want to find. Keep in mind that spaces are treated as characters, and uppercase characters are considered different from lowercase characters (except when `case_off` has been used).

When the string is found, the cursor moves to the first character following it. For example, if you search for the string `bcd` and EMACS finds it in the larger string `abcde`, EMACS moves the cursor to the `e`. Point is after the `d`.

If EMACS cannot find the specified string, the following message appears in the minibuffer:

Not found:

followed by the character string. The position of the cursor does not change.

**Note**

If you do not include the `-NOXOFF` option on your EMACS initialization line, `Ctrl S` acts as the standard PRIMOS stop print command, which freezes the terminal display. To avoid this, you can substitute the `Esc S` command for `Ctrl S`.

**► `abort_command`**`Ctrl G`

The `abort_command` command aborts a partially entered EMACS command. When an EMACS prompt appears in the minibuffer, `Ctrl G` causes the terminal to beep, and the cursor returns to the current file.

▶ **abort\_minibuffer**

Ctrl G

This command is the same as the `abort_command` command.

▶ **all\_modes\_off**

This command turns off all modes in the current buffer. It does not affect modes that you have turned on in other buffers.

▶ **append\_to\_buf**

Ctrl X A

This command deletes the current region and appends it to the buffer you specify. It prompts you for a buffer name. If you specify a buffer that does not exist, EMACS creates a new one for you.

When the region is appended, EMACS displays the following message:

```
Region appended
```

To view the appended text, you must switch to the buffer that contains it.

If you preface this command with a numeric argument other than 1, EMACS copies the region without deleting it.

▶ **append\_to\_file**

Ctrl X Ctrl Z A

This command deletes the current region and appends it to the file you specify. It prompts you for a filename. If you specify a file that does not exist, EMACS creates a new one.

When the region is appended, EMACS displays this message:

```
Region appended
```

If you preface this command with a numeric argument other than 1, EMACS copies the region without deleting it.

▶ **apropos**

Ctrl\_ A

This command displays an extended list of commands and functions that relate to your response to the **Apropos**: minibuffer prompt. For more information about this command, see Chapter 4, Online Help Facility.

▶ **back\_char**

Ctrl B

This command moves the cursor backward one character on the current line. If the cursor is at the beginning of a line, the command moves it across the line separator to the end of the previous line. If the cursor is at the beginning of the buffer, `Ctrl B` does nothing.

► **back\_page** Esc V

This command moves the screen display back one page. The three lines at the top of your current screen move to the bottom of the new screen. The cursor is placed at the beginning of the line in the middle of the display, except at the beginning or end of the file.

You may specify a number of pages to move back by giving a numeric argument to this command.

► **back\_place\_holder** Ctrl X <

This is a speed-type command that moves the cursor to the previous placeholder. The cursor is positioned on the first character in the placeholder. (See Chapter 5, Speed-type, for more information about this command.)

► **back\_tab**

This command moves the cursor back a specified number (indicated by the numeric argument) of tab settings.

► **back\_to\_nonwhite** Esc M

This command moves the cursor back to the first nonwhite character on a line.

► **back\_word** Esc B

This command moves the cursor back and positions it on the first character following the first space it encounters.

► **backward\_clause** Ctrl X Ctrl Z Ctrl A

This command moves the cursor backward the specified number (indicated by the numeric argument) of clauses to the character preceding the last punctuation mark.

► **backward\_kill\_clause** Ctrl X Ctrl Z Ctrl H

This command kills all characters backward from the current cursor position to the last punctuation mark (. ? , or !).

► **backward\_kill\_line** Ctrl X Ctrl K

This command kills all text backward from the current cursor position to the beginning of the line.

▶ **backward\_kill\_sentence****Ctrl X** **Ctrl H**

This command kills text backward from the current cursor position to the last sentence delimiter (. ? or !), or to the beginning of text if no delimiter is encountered.

▶ **backward\_para****Ctrl X** **[**

This command moves the cursor back to the beginning of the paragraph.

▶ **backward\_sentence****Esc** **A**

This command moves the cursor to the character following the previous sentence delimiter (. ? or !).

▶ **begin\_line****Ctrl A**

This command moves the cursor back to the first character of the current line. If the cursor is already at the beginning of a line, it does not move.

▶ **break (PRIMOS command)****Ctrl P**

**Ctrl P** is the PRIMOS break character. It aborts any command in progress, takes you back to PRIMOS command level, and prompts you with the following message:

Control-P typed.

To really Quit from EMACS, type Q

To return to EMACS and Abort the current command (if any), type A

To return to EMACS and Continue with no interruption, type C

Confirm your choice with the Return key.

Typing the Return key without making a choice is the same as Continue.

(If no EMACS command was executing when you typed Control-P, then Abort is the same as Continue.)

You may need to refresh the screen if you choose to re-enter EMACS.

C, A, or Q:

If you type Q to quit from EMACS, any changes that you have not saved during the current session are lost. All responses other than a `Return` or Q, A, or C preceding a `Return` result in the repetition of the last prompt line.

► **capinitial** `Esc` `C`

This command capitalizes the first letter of the current word and puts the remainder of the word in lowercase.

► **case?**

This command prints a message in the minibuffer telling whether case matching is currently enabled.

► **case\_off**

This command turns off case matching during a search. After you type this command, EMACS prints a message in the minibuffer confirming that cases will be ignored when searching.

► **case\_on**

This command turns on case matching during a search. After you type this command, EMACS prints a message in the minibuffer confirming that cases will be looked at when searching.

► **case\_replace?**

This command prints a message in the minibuffer telling you whether case matching is currently enabled for the replacement commands.

► **case\_replace\_off**

This command disables case matching when searching during replace operations.

► **case\_replace\_on**

This command enables case matching when searching during replace operations.

► **center\_line** `Ctrl X` `Ctrl Z` `S`

This command centers the current line of text according to the margins you have set for fill mode. If you type this command with an argument, it centers the number of lines that you specify, beginning with the current one.

▶ **collect\_macro**

Ctrl X (

This command starts defining a keyboard macro. Anything that you type after issuing the **collect\_macro** command becomes part of the macro definition until you issue the **finish\_macro** command.

After you issue this command, the notation **{Macro}** appears on the status line confirming that you are defining a keyboard macro.

▶ **copy\_region**

Esc W

This command places a copy of the current region in the kill ring. It does not delete the region from the buffer or move the cursor.

▶ **cr**

Return

This command moves the cursor and any text after it to the beginning of a new line. If you press **Return** at the end of a line, EMACS moves the cursor to the beginning of a new line. If you press **Return** at the beginning of a line of text, EMACS moves the cursor and the entire line of text down to a new line.

▶ **cret\_indent\_relative**

Ctrl X Return

This command inserts a carriage return into the text, moving the cursor and any text on the line following it to the beginning of a new line. The new line is indented to the first nonwhitespace character of the previous line.

▶ **date**

This extended command inserts the current date into your text at the current cursor position. The date appears in the following format:

```
WED, 10 DEC 1987
```

▶ **default\_tabs**

This command restores tab positions to the default of every five spaces.

▶ **delete\_blank\_lines**

Ctrl X Ctrl O

This command deletes all blank lines immediately following or preceding the current cursor.

**▶ delete\_buffer**

This command deletes all text in the current buffer *without placing it on the kill ring*.

**▶ delete\_char****Ctrl D**

This command deletes the character under the cursor, causing the text on the line to the right of point to shift left. If you type **Ctrl D** at the end of a line, the command deletes the line separator and appends the next line to the current one.

**▶ delete\_region**

This command deletes the current region *without placing it on the kill ring*. Because this command does not save text, be sure to check the boundaries of your region, using **Ctrl X** **Ctrl X** (`exchange_mark`), before issuing it.

**▶ delete\_word****Esc D**

This command deletes the word or partial word following point and places it on the kill ring.

**▶ describe****Ctrl\_ D**

The `describe` command is one of the options of the EMACS online `help` command. See Chapter 4, Online Help Facility, for instructions on using the command.

**▶ display\_buffer**

This command displays information about your current buffer at the top of your screen. The display tells you the name of the buffer, the pathname of the default file, and whether the buffer has been modified. It also tells the current modes that are on and gives mark and cursor information. Any keystroke clears the display.

**▶ display\_debug**

This command displays information at the top of your screen about the debugging facility. Any keystroke clears the display.

**▶ display\_terminal**

This command displays terminal information at the top of your screen. It tells you the terminal type and gives other information such as the height and width of your screen.

**▶ display\_window**

This command gives information about your current screen, such as the line numbers of the top and bottom lines, and the numbers of the far left and far right columns.

**▶ dt**

The **dt** command inserts the date and time into your text at the current cursor position. An example of the format used is shown below:

```
03/12/87 16:35:54
```

The date shows the month first, followed by the day, then the year.

**▶ dump\_file**

This extended command partially compiles the PEEL statements in the current buffer and dumps them in fasdump format to a file with an .EFASL suffix. To make the commands defined in the file available to EMACS, load the partially compiled file with the **load\_compiled** extended command.

**▶ end\_line**Ctrl E

This command moves the cursor to the line separator at the end of the current line. If the cursor is already at the end of a line, it does not move.

**▶ europe\_dt**

This command inserts the date into your current buffer in the following format:

```
12/3/87
```

The day is followed by the month, then the year.

**▶ exchange\_mark**Ctrl X Ctrl X

Ctrl X Ctrl X exchanges the positions of the mark and the point. The point (and the cursor) moves to where the mark was set. The mark moves to the position of point at the time the command was issued.

Since the boundaries of the region remain unchanged when you use this command, you can use it to check the boundaries before you make regional changes. You can also mark a place in text that you want to refer to often from anywhere else in the text. Using Ctrl X Ctrl X as a toggle lets you view the marked area and then return to your current position.

**▶ execute\_macro****Ctrl X E**

This command executes the most recent macro that you have defined during the current editing session. To repeat the execution of this command, you can give it a numeric argument.

**▶ exit\_minibuffer****Return**

This command returns you from the minibuffer to your current cursor position. The command is bound to the Return key in minibuffer mode only.

**▶ expand\_macro**

This command expands your most recent keyboard macro into a function, using the PEEL language. It then prompts you for a name. (If you do not want to give the macro a name, press **Return**.) The command then inserts the source code for this function into the current buffer. You should save the code in a file for later use.

An **expand\_macro** command may result in a series of statements that do not necessarily correspond on a one-to-one basis with the sequence of keystrokes that a user enters from the keyboard.

**▶ explain\_key****Ctrl\_C** or **Esc ?**

This command is an alternate method of invoking the online help facility that explains a keypath. See Chapter 4, Online Help Facility, for more information about this command.

**▶ explore****Ctrl X D**

This command puts explore mode in effect. With explore mode, you can look through directories and read and edit files without leaving EMACS. When you issue the command, EMACS prompts you for the name of the directory you want to explore. If you want to explore your current directory, press **Return**. Otherwise, type the full pathname of the directory you want.

EMACS then switches you to a special "explore" buffer that alphabetically lists first the directories and then the files that are contained in the directory you specified. While you are looking at a directory listing, you can use the explore mode command options that are listed in Table 3-1.

**Table 3-1**  
**Explore Mode Options**

<i>Option</i>	<i>Description</i>
<b>H</b> or <b>?</b>	Displays a list of explore mode commands.
<b>@</b>	Attaches to current explore directory.
<b>A</b>	Displays attributes on screen.
<b>C</b> or <b>N</b>	Creates a file.
<b>D</b> or <b>G</b>	Displays directory or file specified by cursor.
<b>K</b>	Deletes file specified by cursor.
<b>L</b>	Updates information for current directory.
<b>O</b>	Sets spool options for explore mode.
<b>P</b>	Dives into a passworded directory.
<b>Q</b>	Quits and returns you to previous buffer.
<b>R</b>	Renames or relocates file/directory.
<b>S</b>	Spools file specified by cursor.
<b>U</b>	Moves you up a level in directory.
<b>Ctrl X</b> <b>U</b>	Moves you to the next highest directory level from within a file.

If you press **D** while the cursor is positioned on a filename, the contents are displayed. You can edit and save the changes in this text using any of the normal EMACS commands. To return to explore mode from the file text, type **Ctrl X** **U**.

To get out of explore mode, move out of the explore buffer. If you want to exit from EMACS, use any of the interrupt or exit commands.

► **extend\_command**

**Esc** **X**

This command allows you to execute an EMACS command that is not bound to keystrokes by typing **Esc** **X**, followed by the command name. For example,

**Esc** **X** *global\_tabs*

► **fill\_off**

This command turns off fill mode. Fill mode is explained in Chapter 2.

► **fill\_on**

This command turns on fill mode. Fill mode is explained in Chapter 2.

► **fill\_para**

**Esc** **Q**

This command fills and optionally adjusts a paragraph. The filling is done within the constraints of the margins as they are currently set.

A numeric argument greater than 2 to **fill\_para** signals that the paragraph is right-justified as well as filled. An argument of 1 or -1 disables justification. If the argument to this command is less than 0, you cannot create a list type paragraph.

► **find\_file**

**Ctrl X** **Ctrl F**

This command looks for the filename that you specify in response to the prompt. EMACS searches first among EMACS buffers and then among disk files for the filename. When the file is found, EMACS displays the contents on your screen. You may use EMACS editing commands to make changes to the file.

If you try to find a file having the same name (but a different pathname) as one you have already created, found, or read, EMACS tells you that a buffer of that name exists and gives you the pathname of its contents. It prompts you for a new buffer name. If you enter a new name, EMACS creates a new buffer and reads the file into it. (If you foresee this situation, you can create a new buffer ahead of time using **Ctrl X** **B** (**select\_buf**) and then read the file into it using **Ctrl X** **Ctrl R** (**read\_file**). Be aware that if you make changes in the text of this new buffer and save them using **Ctrl X** **Ctrl S** (**save\_file**), EMACS does not write to a new file named after the new buffer, but overwrites the file that you found or read in.

#### Note

If you want to work with a file from another directory requiring a password, you must precede the pathname you specify for **Ctrl X** **Ctrl F** with a tilde (~). For example, if you want to find a file called SECRET in the directory <DSKNAM>CLASSIFIED, requiring the password FISH at the **Find file:** prompt you would type the response in the following format:

```
~<DSKNAM>CLASSIFIED FISH>SECRET
```

Note that the tilde (~) character is placed before the entire pathname, and the password follows the name of the directory (separated by a space) that requires it.

► **finish\_macro** [Ctrl X] ]

This command signals EMACS that you have finished defining a keyboard macro. Any commands that you issue following this command are treated as normal EMACS commands and are not included in the macro definition.

When you issue this command, the **{Macro}** notation disappears from the mode line.

► **forward\_char** [Ctrl F]

[Ctrl F] moves the cursor forward one character position on the current line. If the cursor is at the end of a line, it moves over the line separator to the first character on the next line. If the cursor is at the end of the buffer, [Ctrl F] does nothing.

► **forward\_clause** [Ctrl X] [Ctrl Z] [Ctrl E]

This command moves the cursor forward one clause, leaving it on the character following the first punctuation mark it encounters.

► **forward\_kill\_clause** [Ctrl X] [Ctrl Z] [Ctrl K]

This command kills all characters forward from point to the character following the first punctuation mark it encounters.

► **forward\_kill\_sentence** [Esc] [K]

This command kills all text from point up to and including the first sentence delimiter (. ? or !) it encounters, or to the end of a file if no delimiter is encountered.

► **forward\_para** [Ctrl X] ]

This command moves the cursor forward one paragraph to the first column of the next blank line it encounters. If the cursor is already on a blank line, [Ctrl X] ] moves it to the next blank line.

► **forward\_place\_holder** [Ctrl X] >

This is a speed-type command that moves the cursor forward to the next placeholder in your text. The cursor is positioned on the first character in the placeholder. (See Chapter 5, Speed-type, for more information about this command.)

► **forward\_search\_again**

This command searches forward again for the last string that you specified when you performed a forward search. You can accomplish the same thing using [Ctrl S] and pressing [Return] without entering another string, or by typing [Ctrl C] (**reexecute**) after a search.

▶ **forward\_search\_command****Esc S**

This command is the same as `^s_forward_search_command`.

▶ **forward\_sentence****Esc E**

This command moves the cursor forward and places it on the character following the first sentence delimiter (. ? or !) it encounters.

▶ **forward\_word****Esc F**

This command moves the cursor forward to the first word delimiter (space or punctuation mark) it encounters.

▶ **fundamental**

This command returns you to fundamental mode from any other mode.

- It reestablishes the fundamental mode keybindings for the **Esc** and **Ctrl** keys. The command does not destroy any macros currently defined, but it breaks any bindings between those macros and the **Esc** and **Ctrl** key keypaths.
- It removes the current bindings from all function keys on the terminal, except for keys that send characters identical to fundamental mode keybindings. For example, the PT45 **SEND** key emits **Ctrl W**. After the fundamental command, this key is bound to the `kill_region` macro.

▶ **get\_bufname**

This command inserts the name of your current buffer at point.

▶ **get\_filename****Ctrl X Ctrl Z Ctrl F**

This command inserts the pathname of your current file at point.

▶ **get\_tab**

This command retrieves stored tab stops from a file and puts them into effect. (See the `save_tab` definition in this chapter for information on saving tab stops.) When you issue this command, EMACS asks you for the filename under which the tab stops are stored. As soon as you specify a filename, EMACS switches you to a buffer containing that file and asks you for the name of the function you are using to store the tabs. (The function names are contained in the file being displayed on the screen.)

After you have supplied a function name, EMACS evaluates that function and sets the tab stops you have specified. If you specify a file or function that EMACS cannot find, a **File not found** message appears in the minibuffer and the tabs you requested are not set.

► **global\_tabs**

This command activates global (default) tabs and inserts a message into the minibuffer explaining that global tabs are now in effect.

► **goto\_line**

`[Esc] [G]`

This command moves the cursor to the beginning of a specified line in your buffer. For example, if you type `[Esc] [3] [Esc] [G]`, the cursor moves to the first character on line 3 in your current file. If you do not specify a line number, `[Esc] [G]` moves the cursor to the beginning of line 1 (the top of the buffer).

► **hcol**

This command displays the current setting of the leftmost column of your text display. You can give this command a numeric argument to change the current setting. To set column 50 as the first column, for example, you would type the following:

`[Esc] [5] [0] [Esc] [X] hcol`

In this case, EMACS shifts the display left so that column 50 is positioned at the left edge of the screen and columns 50 through 130 are displayed.

As soon as text is shifted, this message appears:

```
hcol is n
```

where *n* is the new leftmost column number.

When you want to view columns 1 through 80 again, you can reissue this command or you can issue the `reset` command.

If you move point to undisplayed text on either side of the screen, EMACS displays on the status line the number of the column following point, while the cursor remains at the edge of the screen.

► **help\_on\_tap**

`[Ctrl _]`

This command invokes the online EMACS help facility. Chapter 4, Online Help Facility, contains detailed information about online help command options.

► **horiz\_left** [Ctrl X] [ ]

This command shifts the current window left 40 columns and prints a message in the minibuffer telling you which column is currently the leftmost column. Any text to the left of this column is not visible on your screen. If you issue this command when the leftmost column is column 1, EMACS displays the prompt

```
hcol is 1
```

► **horiz\_right** [Ctrl X] [ ]

This command shifts the current window right 40 columns and prints a message in the minibuffer telling which column is currently the leftmost column. If you issue this command when the leftmost column is column 1, after it executes the command, EMACS displays the prompt

```
hcol is 41
```

► **hscroll**

This command tells EMACS to make the column containing point the leftmost column.

► **ignore\_prefix** [Ctrl X] [Ctrl G] or [Ctrl X] [Ctrl Z] [Ctrl G]  
or [Esc] [Ctrl G]

This command aborts a partially typed command. The command is bound to several keypaths so that you can abort commands that are prefixed by different characters.

► **indent\_relative** [Esc] [ ]

This command indents the current line to the same column as the preceding line and positions the cursor on that column.

► **indent\_to\_fill\_prefix** [Esc] [Ctrl I]

This command indents the current line to the left column specified by the `set_left_margin` command. If you do not specify a column number, this command starts the line at column 1.

► **insert\_buf** [Ctrl X] [Ctrl Z] [ ]

This command inserts a copy of the contents of the specified buffer at the current cursor position, subject to the following limitations:

- No more than 32K-1 (32,767) characters may be contained in any 100 contiguous lines of text (that is, the average line length in a 100-line area may not exceed 327 characters).

- The inserted buffer cannot contain more than 3,276,800 lines.
- The command prompts you for a buffer name.

► **insert\_file**

**Ctrl X** **I**

This command inserts a copy of the file you specify in the current buffer at the current cursor position. It prompts you for a filename.

► **insert\_tab**

**Ctrl X** **Ctrl I**

This command inserts spaces from the current cursor position to the next tab stop and moves the cursor to the next tab position.

► **insert\_version**

This command inserts the current EMACS version number in your buffer at the current cursor position. An example is shown below.

```
EMACS version 20.0.12e
```

► **kill\_line**

**Ctrl K**

This command kills all text from the current cursor position to the line separator. If the cursor is on the line separator, **Ctrl K** kills the line separator and joins the following line with the current one. If the cursor is on a blank line containing only the line separator, **Ctrl K** kills the line separator, deleting the blank line. Therefore, if you want to kill a line itself, as well as the text on it, you must type **Ctrl K** twice. EMACS places the deleted text on the kill ring.

► **kill\_region**

**Ctrl W**

This command kills the specified region and places it on the kill ring, moving point back to the mark. (See The Mark and the Region section in Chapter 2 for more information about this command.) Before you issue this command, it is a good idea to check the boundaries of your region, using **Ctrl X** **Ctrl X** (**exchange\_mark**). For information about specifying a region and storing entries on the kill ring, see the **mark** command.

► **kill\_rest\_of\_buffer**

**Esc** **Ctrl D**

This command kills all text from the current cursor position to the end of the current buffer and places the killed text on the kill ring. You can recall the text from the kill ring with **Ctrl Y**.

▶ **leave\_one\_white****Esc** **SPACE**

This command deletes extra spaces preceding the current cursor position, leaving a single blank. If the cursor was on a single space, it moves forward to the next character.

▶ **list\_buffers****Ctrl X** **Ctrl B**

This command displays at the top of your screen a list of all the buffers used during the current editing session in order of creation. This list overwrites the screen display, but it does not affect the text in the buffer. The list disappears as soon as you issue your next command.

**Ctrl X** **Ctrl B** also gives you the following type of information:

```
ref.17 *           3
ref      *         45 <DSKNAM>REF
misc                    115 <DSKNAM>BB>INDOC>$S.9043
.start_up              6 <DSKNAM>EMACS>EMACS_STARTUP
```

The first column in this sample list contains the names of buffers used during the current editing session. Buffer names preceded by a period signify internal buffers used only by EMACS. An asterisk (\*) appearing after a buffer name indicates that the buffer contains unsaved text.

The second column indicates the number of lines in the file. The third column lists the corresponding pathnames of the buffers contained in column 1. If a buffer does not have a corresponding pathname, it means that the buffer does not have an associated disk file. A buffer that does not contain text is not included in the display list.

▶ **load\_compiled**

This command loads a fasload file that has been saved with the **dump\_file** command.

▶ **load\_pl\_source**

This command compiles and loads a PEEL source file. When you type **Esc** **X** **load\_pl\_source**, EMACS prompts you for a pathname. EMACS finds the specified file, compiles it, and then lets you know when it is done. You are then free to execute the commands that are defined in the file.

▶ **local\_tabs**

This command activates tabs that are local to the current buffer. A message appears in the minibuffer saying that local tabs are now in effect.

▶ **lowercase\_region****Ctrl X** **Ctrl L**

This command converts the current region to lowercase. Check the region boundaries before using this command, as corrections can only be made by hand or by rereading the file.

**► lowercase\_word****[Esc] [L]**

This command converts the current word to lowercase and moves the cursor to the space following the word. If you give this command a negative argument, the word or words preceding point will change to lowercase, but the cursor will not move. This is particularly useful when you realize a word is in the wrong case immediately after you have typed it. You can type **[Esc] [1] [Esc] [L]** to change the word and then go on typing without having to move the cursor back and forth.

**► mark****[Ctrl @]**

This command sets a mark at point. The mark defines the beginning of a region. The region ends at the cursor position, after the cursor has moved away from the mark. (See The Mark and the Region section in Chapter 2.)

The mark is also useful for remembering a position in your text. EMACS "remembers" ten previous locations of the mark and stores them on a ring. **[Ctrl U] [Ctrl @]** returns the cursor to previous locations of the mark. If you move to all ten previous locations of the mark, EMACS will bring you back to the most recent one.

**► mark\_bottom****[Ctrl X] [Ctrl Z] [>]**

This command places the mark at the bottom of the current buffer without moving the cursor.

**► mark\_end\_of\_word****[Esc] [@]**

This command sets a mark at the end of the current word without moving the cursor. If the cursor is on a space or punctuation mark, the word following the cursor is the current word. If the cursor is on a character that is already part of a word, that word is the current word.

**► mark\_para****[Esc] [H]**

This command sets a mark at the beginning of the paragraph surrounding point and moves point and the cursor to the end of the paragraph. If the cursor is between paragraphs, **[Esc] [H]** operates on the paragraph preceding the cursor.

**► mark\_top****[Ctrl X] [Ctrl Z] [<]**

This command sets a mark at the top of the current buffer without moving the cursor.

**► mark\_whole****[Ctrl X] [H]**

This command defines the entire buffer as a region by setting a mark at the end of the buffer and moving point and the cursor to the beginning of the buffer.

▶ `merge_lines``Esc` `^`

This command combines the following line with the current one to form a single line in which the text of the joined parts is separated by a single space. Unlike `Ctrl D` or `Ctrl K`, which merge two lines only when the cursor is at the end of a line, `Esc` `^` takes effect regardless of the position of the cursor. The position of the cursor does not change.

▶ `mod_one_window``Ctrl X` `1`

This command displays a single window on your screen. You would normally use this command after you have displayed multiple windows. The window containing the cursor becomes the single window.

▶ `mod_select_buf``Ctrl X` `B`

This command lets you change buffers. It finds or creates the buffer you specify in answer to the prompt, and moves you to it. If you press `Return` after typing `Ctrl X` `B`, EMACS returns you to the previous buffer.

This command is identical to `select_buf`.

▶ `mod_split_window``Ctrl X` `2`

This command divides the current window horizontally into two windows. The top half is window 1 and the bottom half is window 2. Each window displays the contents of a different buffer.

When you issue this command, a line of dashes appears across the middle of your screen to form the two windows. A portion of the text that previously occupied the entire screen remains in window 1. The cursor moves to window 2. The mode line changes to reflect the buffer in window 2, since it contains the cursor.

The first time you type `Ctrl X` `2`, window 2 is automatically given an empty buffer called *alternate*. You can either type text directly into this buffer or use one of the file commands to insert a file into it. If you return to one window and then type `Ctrl X` `2` again during the same editing session, the most recent buffer contained in the other window is automatically displayed in window 2.

Because `Ctrl X` `2` splits the current window into two windows, you can use it to create windows within windows. For example, typing `Ctrl X` `2` with the cursor in window 2 divides window 2 in half to form a total of three windows on your screen.

If you give `Ctrl X` `2` an argument, it makes window 1 only as large as you specify. For example, if you type `Esc` `4` `Ctrl X` `2`, EMACS divides the screen so that window 1 displays only four lines and window 2 displays the remaining lines on the screen.

This feature is very useful when you are working a terminal with a very slow baud rate. If you are working with text in a window made up of only a few lines, you do not have to wait for EMACS to update the entire display after each command.

**► mod\_split\_window\_stay****Ctrl X** **3**

This command is almost identical to **Ctrl X** **2** (**mod\_split\_window**), but it places the cursor in window 1 instead of window 2. The current buffer is not changed, and you are free to continue editing it.

**► mod\_write\_file****Ctrl X** **Ctrl W**

This command writes the text in your buffer to the file listed in the pathname that you specify. If you specify a file that does not exist, EMACS creates it for you. If you specify a file that already contains text, EMACS asks you if you want to overwrite the existing information.

**► move\_bottom****Esc** **>**

This command moves the cursor to the end of the current buffer so that it rests on the last character of the last line. Because PRIMOS inserts a blank line at the end of every file it saves, the last line of the file is blank if the file has been saved. Therefore, the position of the final `cr` in the file determines where this command places the cursor.

**► move\_top****Esc** **<**

This command moves the cursor to the beginning of the current buffer and places it on the first character of the first line.

**► multiplier****Ctrl U**

This command can be used immediately after a numeric argument to multiply the numeric argument by four. The command can only be used when bound to a key sequence normally **Ctrl U**. An example is shown below.

**Esc** **6** **Ctrl U** **x**

This command inserts 24 *x*'s into your buffer. (See the Giving Numeric Arguments to EMACS Commands section in Chapter 1 for more information about **Ctrl U**.)

**► new\_features**

This command displays the buffer (`.new_features`) that contains the file `EMACS*>NEW_FEATURES_INFO`. This file gives information about the current release of EMACS. Type `?` to see a display of options for moving around in this file or returning to your current buffer.

► **next\_buf** [Esc] [N]

This command cycles through all buffers in order of creation following the current one. When you type [Esc] [N], EMACS switches you to the buffer you created immediately after the current one. If the current buffer is the last one you created, [Esc] [N] switches you to "main," the first buffer in the cycle.

When EMACS switches you to a new buffer, you are free to edit that buffer as usual. If you type [Esc] [N] again, EMACS switches you to the next buffer in the cycle. If you type [Esc] [N] enough times, you will eventually get back to your original buffer.

The EMACS Command Conventions section in Chapter 1 explains why [Esc] [N] cannot be used in uppercase letters on the PT200 terminal.

► **next\_line\_command** [Ctrl N]

This command moves the cursor down one line so that it maintains its current column position. If there are no characters (spaces, text, or line separators) in the current column position on the next line, the cursor is positioned following the last used character position in the line. If the cursor is on the last line of your buffer that contains text, [Ctrl N] creates a new line containing no characters and positions the cursor in column 1 of the new line.

► **next\_page** [Ctrl V]

This command moves the screen display forward one page. The last three lines at the bottom of your screen move to the top of the new screen display. The cursor is placed at the beginning of the line in the middle of the display, except at the beginning or end of the file.

You may specify a number of pages to move forward by giving a numeric argument to this command.

► **one\_window**

This command is analogous to the `mod_one_window` command described above.

► **open\_line** [Ctrl O]

This command is almost identical to `cr`, described above, but it maintains the current position of the cursor.

If you type [Ctrl O] at the end of a line, EMACS inserts a carriage return after the cursor and the cursor remains where it is. If you type [Ctrl O] in the middle of a line of text, EMACS inserts a `cr` after the cursor, moves the text following the cursor to new line, and leaves the cursor in its present position. If you type [Ctrl O] at the beginning of a line, EMACS inserts a `cr` after the cursor, moves the entire line of text down to a new line, and leaves the cursor at its present position.

**▶ other\_window****Ctrl X** **O**

This command moves the cursor to the other window. If there are more than two windows, **Ctrl X** **O** moves the cursor back and forth between the last two windows you created.

**▶ overlayer**

This function takes effect only within overlay mode. It overlays the existing character with a new one. It is similar to the `self_insert` function in fundamental mode and cannot be executed as an extended command.

**▶ overlay\_off**

This command turns off overlay mode so that newly entered characters are inserted at point.

**▶ overlay\_on**

This command puts overlay mode in effect until you turn it off. In overlay mode, EMACS deletes any existing character on the line at point before adding a new one at the same position.

**▶ overlay\_rubout**

In overlay mode, this command replaces the character to the left of the cursor with a space and moves the cursor back to the space.

**▶ pl**

This command compiles the Prime EMACS Extension Language source code in your current buffer so that you can execute it during the current EMACS session. You can name another buffer as an argument. If the buffer contains more than 32,000 characters, the command produces an error message.

**▶ pl\_minibuffer****Esc** **Esc**

This command allows you to execute a PEEL statement. EMACS displays the following prompt in the minibuffer:

PL:

Respond to the prompt with a PEEL statement for EMACS to execute.

**► popmark**

This command "pops" the current mark off the mark ring. It does not change the cursor.

**► prepend\_to\_buf****Ctrl X P**

This command inserts the current region at the beginning of the buffer you specify. It prompts you for a buffer name. If you specify a nonexistent buffer, EMACS creates a new buffer for you.

When the region is prepended, EMACS displays the following message:

```
Region prepended
```

If you do not specify a numeric argument greater than 1, the current region is deleted and moved to the new buffer. If you specify a numeric argument greater than 1, the region is copied and moved.

**► prepend\_to\_file****Ctrl X Ctrl Z P**

This command inserts the current region at the beginning of the specified file. The command prompts you for a filename. If you specify a nonexistent file, EMACS creates a new file for you.

When the region has been prepended, EMACS displays the following message in the minibuffer:

```
Region prepended
```

If you do not specify a numeric argument greater than 1, the region is deleted and moved to the file. If you specify a numeric argument greater than 1, the region is copied and moved.

**► prev\_buf****Esc P**

This command cycles through all of the buffers in reverse order of creation, from the current one.

**► prev\_line\_command****Ctrl Z**

**Ctrl Z** moves the cursor up one line so that it maintains its current column position. If there are no characters (spaces, text, or line separators) in the current column position on the previous line, the cursor is positioned following the last used character position in the line.

If you type **Ctrl Z** on the first line of a file, EMACS ignores the command.

## ► `primos_command`

Ctrl X Ctrl E

Ctrl X Ctrl E allows you to execute a PRIMOS command without leaving EMACS. When you issue this command, EMACS displays the following prompt in the minibuffer:

Primos command:

You can respond to this prompt in one of three ways:

- Type one exclamation point (!) before the command.
- Type two exclamation points (!! ) before the command. This is the least complex method.
- Do not precede the command with any characters.

Each method is discussed below.

If you precede the PRIMOS command with one exclamation point (!), EMACS executes the command and switches you to a new buffer called `file_output` to display the output. You can modify and save the text in this output buffer just like any ordinary EMACS buffer. To get back to your original buffer, use any of the buffer or file commands.

If you precede the PRIMOS command with two exclamation points (!! ), EMACS executes the command, but it does not switch you to a different buffer. Instead, EMACS generates a new screen, runs the command at PRIMOS level, and displays the output on that screen. This output disappears when you type any single character, or a single command character such as Ctrl L or Ctrl G; you cannot save it. Note that if you type a single character, EMACS inserts that character into your current file.

You may use one of your PRIMOS abbreviations in response to the **Primos command:** prompt by typing either one or two exclamation points, followed by `ab -ee` and your abbreviation. The following example shows how to use an abbreviation `d` (for the PRIMOS delete command) to delete a file from your current directory.

Primos command: `!!ab -ee d pathname`

If you issue a PRIMOS command without any characters preceding it, EMACS executes the command as a phantom process. When the command is executed, EMACS switches you to the `file_output` buffer and displays the output of this command.

The names of these three options to the `primos_command` command are shown in Table 3-2.

**Table 3-2**  
Options to `primos_command`

<i>Option</i>	<i>Form</i>	<i>Description</i>
<code>primos_internal_como</code>	!	Output is displayed in <code>file_output</code> buffer.
<code>primos_internal_screen</code>	!!	Output is displayed on new screen.
<code>primos_external</code>	no prefix	Command is executed as phantom.

## Notes

At Rev. 19.4 of PRIMOS, any PRIMOS command except EMACS may be executed via `primos_internal_como` or `primos_internal_screen`. External commands no longer overwrite EMACS.

It is not possible to execute a .COMI file from within EMACS, using either `primos_internal_como` or `primos_internal_screen`. An attempt to do so will result in the following message:

```
Use CPL (instead of COMINPUT) to execute a file
containing PRIMOS commands!
```

A .COMI file can be executed, however, using `primos_external`.

### ► `primos_external`

The `primos_external` command or function is the option to the `[Ctrl X] [Ctrl E]` command that executes a PRIMOS command by means of a separate phantom job. It waits for the execution to complete and then displays the results in the `file_output` buffer.

After the phantom job has terminated, EMACS creates or overwrites the text buffer named `file_output`, loading into it the output file created by the phantom job. You may continue editing your original file by switching back to the buffer into which that file was loaded.

### ► `primos_internal_como`

The `primos_internal_como` command or function is the option to the `[Ctrl X] [Ctrl E]` command that runs a PRIMOS command with `como` output. If the command is not specified, EMACS prompts you with **Internal command:**. The PRIMOS command must not require any interactive keyboard input from the user.

After execution of the command is completed, EMACS creates or overwrites the text buffer named `file_output`, loading into it the command output file. Screen displays normally generated by the PRIMOS command are not shown on the screen, but are captured in the output file.

### ► `primos_internal_screen` function

The `primos_internal_screen` function is the option to the `[Ctrl X] [Ctrl E]` command (!! ) that executes an interactive PRIMOS command. The following prompt appears in the minibuffer:

```
Internal command:
```

When you enter the PRIMOS command, EMACS clears your current screen and displays the command output on your screen. To return to your original screen, type any single-character command such as `[Ctrl G]`, or any single character.

**► pushmark**

This command pushes a mark onto the ring of marks. The ring contains ten entries.

**► query\_replace****Esc %**

This command searches for a character string with the option of replacing it with another. EMACS prompts you for each instance. EMACS finds the first occurrence of the first string and places the cursor on the character following it. It then waits for you to respond to the following prompt:

SPACE = Replace+Continue CR = Skip+Continue "." = Replace+Stop ^G = Abort

The responses work as follows:

<i>Response</i>	<i>Description</i>
<b>SPACE</b>	Replaces current occurrence of the first string and then moves to next one.
<b>Return</b>	Skips to next occurrence of the first string without replacing the current one.
<b>.</b>	Replaces current occurrence of string and exits without doing any more replacement.
<b>Ctrl G</b>	Stops searching and does no more replacements.

EMACS ignores any other character you type until the search is stopped. When the search is stopped, the cursor moves back to its original position.

The **query\_replace** command works only within the current region, if one is defined. If **query\_replace** fails to find an instance of your first string where you are sure one exists, check the boundaries of your region.

**► quit****Ctrl X Ctrl C**

This command takes you out of the EMACS editor and back to the process from which you invoked EMACS, normally PRIMOS command level. If any buffers contain text that has not been saved, EMACS prints a list of these buffers at the top of your screen and asks:

```
Above list of modified buffers(s) not saved to files(s).  
Quit anyway?:
```

If you respond by typing Y or YES, you leave EMACS and your changes are lost. If you type N or NO or **Ctrl G**, EMACS aborts the request to exit.

▶ **quote\_command**

Ctrl X Q

This command is the same as the `^q_quote_command` command.

▶ **read\_file**

Ctrl X Ctrl R

This command copies an existing PRIMOS disk file into your current EMACS buffer and displays it on your screen. It prompts you for a filename.

If you specify a file that does not exist, EMACS responds with an error message. If your buffer already contains text, EMACS informs you that the buffer is not empty and asks if you want to delete it. If you give a positive response, EMACS deletes the old text and inserts the new file in its place. If you give a negative response, EMACS ignores your request to read in a file.

When you read a file into a buffer using `Ctrl X Ctrl R`, the name of the buffer does not change.

**Note**

If you want to work with a file from another directory requiring a password, you must precede the pathname you specify for `Ctrl X Ctrl R` with a tilde (~). For example, if you want to find a file called SECRET in the directory <DSKNAM>CLASSIFIED, requiring the password FISH, you would type the following string in response to the **Read file** prompt:

```
~<DSKNAM>CLASSIFIED FISH>SECRET
```

Note that the tilde (~) character is placed before the entire pathname, and the password follows immediately after the name of the directory (separated by a space) that requires it.

▶ **reexecute**

Ctrl C

This command reexecutes the last command you issued. It is particularly useful when you want to reissue a command that requires many keystrokes. If the last command you issued required a response to a prompt, EMACS remembers your response. If you issue `Ctrl C` while you are entering text, `Ctrl C` reenters the last character you typed.

▶ **refresh**

Ctrl L

This command refreshes the text on your screen by removing all text that is not part of the text you are editing, such as messages from the system or information resulting from certain EMACS commands. This command also centers point in the middle of the screen.

`Ctrl L` with a positive numeric argument moves the text so that the cursor is on the line specified by the argument. For example, an argument of 3 moves the text so that the cursor is on line 3, counting down from the top of the screen.

**Ctrl L** with an argument of 0 moves the text so that the cursor is on the top line of the screen.

**Ctrl L** with a negative numeric argument moves the text so that the cursor is on the line specified by the argument, counting up from the bottom of the screen. For example, giving this command an argument of -2 moves the text so that the cursor is on the second line from the bottom of the screen.

► **reject**

This command prints the **Invalid command:** prompt in the minibuffer. See the *EMACS Extension Writing Guide* for more information.

► **repaint**

**Ctrl X R**

This command moves your cursor to the first line on your screen. You can move the cursor to any line that you specify by giving a positive numeric argument. If you use a negative or 0 argument, EMACS assumes that you want the cursor moved to line 1.

► **replace**

This command replaces all occurrences of one character string with another. EMACS prompts you in the minibuffer for both character strings. If you create a keyboard macro using the Global Replace key, EMACS is inconsistent in retaining the search and replace strings within the macro. (See the section on Macros in Chapter 2.)

The **replace** command works only within the current region if one is defined. If **Esc X replace** fails to find an instance of your first string where you are sure one exists, check the boundaries of your region.

► **reset**

This command refreshes the screen and makes column 1 the leftmost column. It is a general reset command in that it turns off any modes or special features you have in effect such as multiple windows.

► **reverse\_search\_again**

This command searches backward again for the last string that you specified when you performed a reverse search. The position of the cursor and the minibuffer prompts are the same as for a reverse search.

▶ `reverse_search_command``Ctrl R` or `Esc R`

This command searches backward from the cursor for the character string you specify. When the search is successful, the cursor moves to the first character in the string. Therefore, if you search for the string `bcd` and EMACS finds it in the larger string `abcde`, EMACS places the cursor on the `b`. If the search is unsuccessful, the `Not found:` prompt appears in the minibuffer.

▶ `rubout_char``Ctrl H`

`Ctrl H` deletes the character before point, causing the remaining text on the line to shift left. If you type `Ctrl H` at the beginning of a line, it deletes the line separator character occurring before it and causes the current line and the previous one to join at the end of the previous line.

Most terminals have special keys that perform the same function as `Ctrl H`. These are:

`Backspace` `Delete`▶ `rubout_word``Esc` `Ctrl H`

This command deletes the previous word and places it on the kill ring. If you press `Esc` `Backspace` in the middle of a word, only those characters preceding the cursor are deleted.

You can usually substitute `Delete` or `RUBOUT` for `Backspace` in this command. To check this, type `Ctrl _ C` (`help_on_tap`) followed by the keystrokes. This command tells you if those keystrokes invoke the `rubout_word` command.

▶ `save_all_files`

This command saves all modified buffers that have associated disk files. It returns a message telling how many files were saved.

▶ `save_file``Ctrl X` `Ctrl S` or `Ctrl X S`

This command saves the text from your current buffer and writes it to the filename specified on the status line. You can issue this command at any time during an editing session as long as a filename is listed on the status line. If you try to use `Ctrl X` `Ctrl S` in a buffer that has no corresponding filename, EMACS responds with the error message:

```
No default file name for this buffer
```

To save text from a buffer that has no filename, use the `Ctrl X` `Ctrl W` (`mod_write_file`) command.

**► save\_tab**

This command saves the current tab stops in a file so you can retrieve them at any time. The tab stops are actually stored in an EMACS function, which is in turn stored in a file. Therefore, you can save as many sets of tab stops as you like in one file as long as they are stored under different function names.

When you type **[Esc] [X] save\_tab**, EMACS prompts you for a filename. If you specify an existing file, EMACS switches you to a buffer containing that file. If you specify a new file, EMACS creates it for you and switches you to an empty buffer. You are then prompted for a name under which to save the tabs. This is the name of the function. When you supply a name, the source code for the function is inserted into the file, you are returned to your original text, and this message appears in the minibuffer:

```
Tabs are now saved
```

**► scroll\_other\_backward****[Ctrl X] [V]**

This command moves the text in the other window backward one screen without moving the cursor from the current window. The other window is the next-last created window, if you are displaying more than two.

If you give this command a positive argument, it moves the text backward the specified number of lines. If you give this command a single minus sign (-) for an argument, it moves the text forward one screen. If you give this command a negative argument, it moves the text forward the specified number of lines.

**► scroll\_other\_forward****[Esc] [Ctrl V]**

This command moves the text in the other window forward one screen without moving the cursor out of the current window. The other window is the next-last created window, if you are displaying more than two.

If you give this command a positive argument, it moves the text forward the specified number of lines. If you give this command a single minus sign (-) for an argument, it moves the text backward one screen. If you give this command a negative argument, it moves the text backward the specified number of lines.

**► select\_any\_window****[Ctrl X] [4]**

This command cycles the cursor through all windows in the reverse order of their creation.

**► select\_buf**

This command is the same as **mod\_select\_buf**.

### ► `self_insert`

This function inserts a character into your text buffer. It is normally bound to all printing keys, that is, keys for printing characters not preceded by an `[Esc]` or a `[Ctrl]` character. Therefore, to insert text in a file, all you have to do is type the text. As you insert characters, point and any characters following it move to the right on the line. For example, if point is located between the characters `o` and `a` in the word `infoation`, typing `rm` produces `information`, with the cursor on the `a`.

This function cannot be executed as an extended command. See the Functions section in Chapter 1 for an example of executing `self_insert` as a PI function.

### ► `set_hscroll`

This command sets the value of `hcol`, the first column of text on your screen. It prompts you for the column number. After you enter this number, EMACS shifts your screen display left until that column is displayed at the left edge of the screen. Any text to the left of this column is shifted off the screen. Text to the right of that column not previously visible shifts into view. For example, if you specify the left column to be 50, the text in columns 50 through 130 will be displayed on the screen. You are free to edit the shifted text with any of the normal EMACS commands.

As soon as text is shifted, this message is displayed in the minibuffer:

```
hcol is n
```

where  $n$  is the leftmost column number.

When you want to view columns 1 through 80 again, you can reissue this command and specify column 1 when prompted for a number, or you can issue the `reset` command.

If you move point to undisplayed text on either side of the screen, EMACS displays on the status line the number of the column following point, while the cursor remains at the edge of the screen.

### ► `set_key`

This command is similar to `[Esc] [X] set_permanent_key`, except that the keybindings you create are effective in the current buffer only. When you switch to a new buffer, the old keybindings remain in effect.

### ► `set_left_margin`

This command sets the left fill margin to the column you specify. EMACS prompts you for the column number.

### ► `set_mode`

This command turns off any other mode in your current buffer and turns on the mode you specify. EMACS prompts you for a mode name. The new mode name appears on the status line.

► **set\_mode\_key**

This command binds a keypath to a function in the specified mode. EMACS prompts you for a mode name, then for a keypath, and finally for the name of a function. This command is similar to the **[Esc] [X]** `set_permanent_key` command except that the keybindings are effective only in buffers where the mode you specified is in effect.

► **set\_permanent\_key**

This command binds a function to the key you specify for the duration of the current editing session. It prompts you for a keypath and a command name. The keypath refers to the keys to which you want the function bound. The command name is the name of the function you want to bind to a key.

**Note**

You cannot use **[Ctrl M]** or **[Ctrl m]** to bind an EMACS command. PRIMOS changes the terminal's carriage return character (**[Ctrl M]**) to **[Ctrl J]**, and completely ignores the terminal's line-feed character (**[Ctrl J]**). Thus, PRIMOS prevents a **[Ctrl M]** from ever reaching EMACS.

To specify the keypath, type printing characters that correspond to the keys you want to use. The keypath may not exceed 10 characters. The conventions for typing keypaths are as follows:

^ means **[Ctrl]**

^[ means **[Esc]**

**Note**

When you bind a function to an **[Esc]** sequence, you must bind uppercase and lowercase letters separately. For instance, if you want to bind a function to **[Esc] [Q]** and to **[Esc] [q]**, you must issue the **[Esc] [X]** `set_permanent_key` command twice: once to set the keypath to **^[Q** and once to set it to **^[q**.

The new bindings remain in effect for the duration of the current editing session. If you want to use these bindings every time you use EMACS, you must incorporate them into an EMACS library. The *EMACS Extension Writing Guide* and Chapter 6 of this book explain how to do this.

► **set\_right\_margin**

**[Ctrl X] [F]**

This command sets the right fill margin to the column you specify. EMACS prompts you for a column number.

▶ **set\_tab**

This command is an alternate for the **settab** command.

▶ **set\_tabs**

This command is an alternate for the **settab** command.

▶ **setmark**

This command is identical to the **mark** command.

▶ **settab**

This command allows you to set tab stops in any columns you like. When you issue this command, EMACS switches you to another buffer containing an 80-character ruler at the top of the screen. The following prompts appear in the minibuffer:

Is there a default interval:

If you type

**yes**

EMACS asks you

How far apart:

You can then type a number to specify a new default interval. When you do, point moves to column 1 on the ruler at the top of the screen and a T appears in each column that contains a tab stop. For example, if you specified a default interval of 8, the ruler would look like this:

```

          1           2           3           4           5           6           7           8
...5...0...5...0...5...0...5...0...5...0...5...0...5...0...5...0...5...0...5...0...5...0
          T           T           T           T           T           T           T           T

```

When you have set a default interval, you are free to modify it with the tab commands described in Table 3-3.

If you do not specify a default interval, the cursor moves immediately to column 1 on the ruler and a T appears in column 80.

When the cursor moves to the top of your screen, the following prompt appears in the minibuffer:

Type a space, t, b, h, r, ?, or q

These options are summarized in Table 3-3.

**Table 3-3**  
**Settab Prompts**

<i>Prompt</i>	<i>Explanation</i>
<b>SPACE</b>	Moves cursor forward one character on line under ruler. Deletes a T.
<b>t</b>	Sets a tab stop at specified column.
<b>f</b>	Moves the cursor forward one character on line under ruler. Does not delete a T.
<b>b</b>	Moves cursor backward one character on line under ruler. Does not delete a T.
<b>h</b>	Shifts ruler left 60 spaces. Allows you to set tab stops in columns you normally do not see on the screen.
<b>r</b>	Resets ruler to original position.
<b>q</b>	Signals EMACS that you want to quit setting tab stops and go back to your text.
<b>?</b>	Describes the list of tab command options.

#### ► **settabs\_from\_table**

This command sets tab stops at the first column of every word on the line where the cursor is. (A word is a sequence of consecutive characters delimited by a space or punctuation mark.)

For example, suppose the cursor is somewhere on a line containing the text shown below:

```
COLUMN_A      B          123          COLUMN D
```

EMACS sets tab stops at the following character positions:

- On the C in the word COLUMN\_A
- On the B
- On the 1 in the character sequence 123
- On the C in the word COLUMN
- On the D

Note that COLUMN\_A is considered one word and COLUMN D is considered two words.

You may abbreviate this command to **setft**.

**▶ set\_user\_type**

This command sets the user type for the file hook procedure.

**▶ sort\_dt**

This command inserts the date into your buffer in the following format:

```
87/12/17
```

**▶ spd\_add**

This command adds an abbreviation to the current speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_add\_modal**

This command adds a speed-type abbreviation that is defined for a specific mode. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_add\_region**Ctrl X +

This command defines a region in your text as the expansion of a speed-type abbreviation. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_compile**

This command compiles the speed-type source file in the current buffer. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_delete**

This command deletes an existing abbreviation from the current speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_list**

This command prints information about a specific speed-type abbreviation at the top of your screen. For detailed information about how to use speed-type, see Chapter 5.

**▶ spd\_list\_all**

This command gives you information about all the abbreviations currently loaded into the speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_list\_file**

This command lists the speed-type abbreviations found in the file that you specify. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_load\_file**

This command loads an abbreviation file into the speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_off**

This command turns off the speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_on**

This command turns on the speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_save\_file**

This command saves all the changes you have made to the current speed-type environment. For detailed information about how to use speed-type, see Chapter 5.

▶ **spd\_unexpand**

Ctrl X -

This command removes the most recent speed-type expansion from the current buffer. For detailed information about how to use speed-type, see Chapter 5.

▶ **split\_line**

Esc Ctrl O

This command breaks a line at the cursor. The characters to the right of the cursor move down one line to the same column positions on the new line. The cursor does not move.

▶ **split\_window**

This command is the same as `mod_split_window`.

▶ **split\_window\_stay**

This command is the same as the `mod_split_window_stay` command.

▶ **tablist**

This command sets tab stops from a list of column numbers that you supply. When you issue this command, the following prompt appears:

Set tab columns separated by blanks:

Enter the column numbers where you want tab stops, leaving spaces between them. This message then appears in the minibuffer:

Tabs are set

▶ **take\_left\_margin**

**Ctrl X** **.**

This command sets the left fill margin to the column currently containing the cursor. It is effective only if it is followed by the **fill\_para** command. A minibuffer message displays the column number of the new left margin.

The **untidy** command restores text in filled but unjustified format, using column 1 as the left margin. (Resetting the margin and using the **fill\_para** command does not give the same result, due to the introduced spaces.)

▶ **take\_right\_margin**

**Ctrl X** **Ctrl Z** **F**

This command sets the right fill margin to the column currently containing point. A minibuffer message displays the column number of the right margin. This command sets a new right margin when you are executing the **fill\_para** command.

▶ **tell\_left\_margin**

This command displays in the minibuffer the column number of the current left margin.

▶ **tell\_modes**

This command gives information at the top of the screen about all modes that are currently in effect. The command is useful when modes are in effect whose names are not on the status line. You can clear the display with **Ctrl G** (**abort\_command**).

▶ **tell\_position**

**Ctrl X** **=**

This command displays information in the minibuffer about the current buffer, including the current line and character position. A sample line of information looks like this:

At line 26 of 240, col 26 char is 254 (,). Window line 13.

**At line 26 of 240** means that the cursor is currently on line 26 of a file that is 240 lines long.

**Col 26** means that the cursor is currently in column 26 on the screen.

**Char is 254** refers to the octal code number of the character after point (under the cursor). The comma in parentheses is the actual character.

**Window line 13** means that point is on line 13 of the terminal screen.

▶ **tell\_right\_margin**

This command displays in the minibuffer the column number of the current right margin.

▶ **toggle\_redisplay**

`Ctrl X` `Ctrl T`

This command freezes the screen display of an EMACS command until you issue it again. It is usually used with slow display terminals.

While the screen is frozen, you can issue EMACS commands, and EMACS will execute them. If you want to check on the execution, you can issue the `view_lines` command, and EMACS will display a single screen showing the current activity.

If you issue an invalid command, EMACS displays an error message in the minibuffer and displays the current state of your screen. Issuing `Ctrl X` `Ctrl T` again unfreezes the screen display.

▶ **transpose\_word**

`Esc` `T`

This command transposes the two words immediately before and after the cursor. If the cursor is in the middle of a word, `Esc` `T` transposes that word and the one preceding it. The cursor goes to the first character of the second word.

▶ **trim\_date**

This command inserts the current date into your buffer at the current cursor position in the following format:

```
18 Dec 1987
```

▶ **trim\_dt**

This command inserts the current date into your buffer at the current cursor position in the following format:

```
12/18/87
```

▶ **twiddle**

Ctrl T

This command transposes the two characters immediately preceding point. For example, if the cursor rests on the **e** in the character string **ht**e****, typing **Ctrl T** transposes the **h** and **t** to form the word **the**. The cursor does not move.

▶ **type\_tab**

Ctrl I

This command works like a typewriter Tab key. It moves point forward on a line to the first tab stop it encounters.

If your terminal has a **TAB** key, you can press it instead of typing **Esc X type\_tab**.

This command takes both positive and negative arguments. A negative argument causes this command to act as a backtab.

▶ **unmodify**

Esc ~

This command tells EMACS to treat the current buffer as if it were unmodified. It clears the modified flag and allows you to quit EMACS without being warned that you made changes and did not save them.

▶ **untidy**

This command reformats a paragraph so that the lines are about the same length. It removes indentation and justification while filling the paragraph. It uses column 1 for the left margin and the latest setting for the right margin.

▶ **uppercase\_region**

Ctrl X Ctrl U

This command converts the current region to uppercase. Check the region boundaries before using this command, as corrections can only be made by hand or by rereading the file.

▶ **uppercase\_word**

Esc U

This command converts the current word to uppercase and moves point to the space following the word.

▶ **view**

This command turns on view mode in the current buffer. When view mode is in effect, the current buffer cannot be modified. You can use the options listed in Table 3-4, as well as your function keys, to move through the buffer.

**Table 3-4**  
**View Mode Options**

<i>Option</i>	<i>Description</i>
<b>SPACE</b>	Moves cursor forward a screen.
<b>B</b>	Moves cursor back one screen.
<b>&gt;</b>	Moves cursor to start of text.
<b>&lt;</b>	Moves cursor to end of text.
<b>R</b>	Executes reverse search.
<b>S</b>	Executes forward search.

► **view\_file**

**Ctrl X** **Ctrl V**

This command finds the file that you specify in response to the prompt, reads it into a buffer, and switches you to that buffer in view mode. You can then use view mode options to page through the text. (See the `view` command for a list of view mode options.) The minibuffer contains instructions for turning view mode off.

► **view\_kill\_ring**

**Ctrl X** **Ctrl Z** **K**

This command switches you from your current buffer to the buffer containing the kill ring. This buffer displays one kill ring entry on the screen. If the kill ring is not full or one of your kills was just blank lines, the screen may be blank and a message appears in the minibuffer. The prompt

View Text: Type s, n, v, q, ?

appears in the minibuffer. These options are described in Table 3-5.

**Table 3-5**  
**Kill Ring Options**

<i>Option</i>	<i>Description</i>
<b>s</b>	Saves the current entry so that it can be yanked back immediately with <b>Ctrl X</b> <b>Ctrl Z</b> <b>Ctrl Y</b> ( <code>yank_kill_text</code> ), described below.
<b>n</b>	Displays the next kill ring entry on the screen. After the display of the last kill ring entry, you return to your original buffer; there is no message, but the status line reflects the return.
<b>v</b>	Allows you to view text.
<b>q</b>	Quits displaying kill ring entries and takes you back to your original buffer.

The first entry you view on the kill ring is not necessarily the first or most recent text you killed. **Ctrl X Ctrl Z K** shows you an arbitrary entry and then displays subsequent entries in order on the ring.

If you give this command a numeric argument other than 1, EMACS displays the contents of a special text kill ring. This kill ring works just like the standard kill ring and is used to save any changes made as a result of the following commands: (1) **Ctrl X Ctrl U** (**uppercase\_region**), (2) **Ctrl X Ctrl L** (**lowercase\_region**), (3) **Esc Q** (**fill\_para**), and (4) **untidy**. In the case of the latter two commands, the contents of temporary, intermediate buffers created during the execution of the commands are maintained on the kill ring.

► **view\_lines**

**Ctrl X Ctrl Z Ctrl V**

This command is used with the **toggle\_redisplay** command. It displays one screen showing the current EMACS activity, after which the screen display remains frozen.

► **view\_off**

**Ctrl U Ctrl X Ctrl V**

This command turns off view mode in the current buffer.

► **view\_on**

This command is identical to **view**.

► **vsplit**

This command splits your screen vertically at point, if point is at least 10 characters from the left margin. The cursor moves to the new window. The new window contains a display of the current buffer until you read in a new buffer or file.

Any of the window commands that apply to horizontal windows apply to vertical windows, and you may create more than two vertical windows. If you want to shift the window display horizontally, you can use the **horiz\_left** or **horiz\_right** commands.

► **wallpaper**

This command inserts a complete list of current EMACS commands and functions into your buffer.

► **which\_tabs**

This command inserts a message in the minibuffer telling which tabs are now in effect.

▶ **white\_delete****Esc** **\**

This command deletes whitespaces (nonprinting characters) as follows:

- If the cursor rests on a character, EMACS deletes whitespace immediately preceding it and adjusts any remaining text on the line to the left. The cursor moves with the character it started on.
- If the cursor rests on a space, EMACS deletes whitespace immediately on either side and adjusts any remaining text on the line to the left. The cursor moves to the first character that followed the whitespace.

▶ **wrap**

This command is bound to the **SPACE** key and **Return** in fill mode. It checks to see whether the last character on a line extends beyond the fill margin and inserts a new line if it does.

▶ **write\_file**

This command saves the text in the current buffer in the file that you specify in response to the prompt. By writing to different filenames, you can save an original file and all of its edited versions. This command overwrites an existing file with no warning message.

▶ **yank\_kill\_text****Ctrl X** **Ctrl Z** **Ctrl Y**

This command yanks the text saved during **view\_kill\_ring** and reinserts it in your buffer at point.

▶ **yank\_minibuffer****Esc** **Ctrl Y**

This command inserts the characters you typed as a previous minibuffer response into your buffer at the current cursor position. It is most useful within the minibuffer prompt to retype a previous response.

▶ **yank\_region****Ctrl Y**

This command yanks the most recently killed text from the kill ring and places it in the buffer at point. It moves point to the end of the reinserted text and places the mark at the beginning of the inserted text. Note that you can yank the same block of text from the kill ring as many times as you like.

▶ **yank\_replace****Esc** **Y**

This command recalls text from the kill ring, cycling through all the entries (including empty ones) on the ring as the command is reissued. If you have not moved the cursor from its position after a previous yank, **Esc** **Y** replaces the just recalled text with the next latest ring entry. If you have moved the cursor, text is inserted at point.

---

# 4

## Online Help Facility

### Introduction

This chapter describes the EMACS online help facility that you can access with the `Ctrl_` (control underscore) command. The chapter explains how to decide which help command option to use and how the options work.

### EMACS Help Command

EMACS has an online help facility that you can use at any time during an editing session. To find out which help command option is appropriate for your task, type the `Ctrl_` command. EMACS responds with the following prompt that provides a list of the help command options.

Help on tap: C=explain key, A=apropos, D=describe, L=last 20 chars, ?=more help

The help command is defined as follows:

<i>Keybinding</i>	<i>Command Name</i>	<i>Description</i>
<code>Ctrl_</code>	<code>help_on_tap</code>	Lists options for the help command.

The help command options and brief explanations of them are as follows:

<i>Option</i>	<i>Command Name</i>	<i>Description</i>
A	<code>apropos</code>	Lists all commands related to an operation.
C	<code>explain_key</code>	Lists what a keystroke does.
D	<code>describe</code>	Gives information about commands and PEEL statements.
L		Lists the last 20 characters you typed.
?		Lists the help command options.

### Note

You can invoke any of the help command options directly by typing the option you want right after the main command without waiting for the prompt. For example, `Ctrl-L` immediately lists the last 20 characters you typed. Options can be typed in either uppercase or lowercase characters.

The following sections discuss each option.

## Apropos

The Apropos option helps you determine which EMACS commands to use for a particular action by giving you a list of possible commands for that action. For example, if you want information about deleting, you would type `Ctrl-A`.

EMACS prints the **Apropos:** prompt in the minibuffer and waits for you to respond with a word that specifies a subject. You might type the word **delete** in response to the prompt, as follows:

```
Apropos: delete
```

In this case, EMACS displays a list of commands containing the word **delete** at the top of your screen, separated from your current text by a double row of dashes. If EMACS cannot find information about your command, it responds with only the dashes. In that case, try another word related to the subject, or even a portion of your original word. You must use ingenuity when you specify subjects. For example, if you are interested in commands that kill backward and the string **backward\_kill** does not reveal any, do not give up. Try typing **kill** or **backward** or just **back**. Be persistent. To clear the Apropos display and return to your current file, type `Ctrl-G` (or any character).

You can also invoke the Apropos option by typing `Esc X apropos`.

## Explain\_Key

The C option explains what a particular character command does. (C stands for *character*.)

When you type `Ctrl-C`, EMACS responds with the prompt:

```
Explain key:
```

in the minibuffer. It then waits for you to type in a sequence of characters that make up a command and responds with an explanation of what that command does. For instance, if you type `Ctrl-F` after the **Explain key:** prompt, EMACS responds with the explanation:

```
forward_char: Forward character
```

Keep in mind that because the C option explains only the first command you type after the **Explain key:** prompt, you cannot use it to explain extended commands. If you type an extended command, you will get an explanation of `[Esc] [X]`, which was the first character command you typed.

If you type keystrokes that EMACS does not recognize, EMACS prompts you with the following error message:

```
Undefined key
```

You can also invoke the C option by typing `[Esc] [?]`.

If you need a more detailed description of a command, you can use the D option.

## Describe

The D option describes EMACS commands and functions in detail. (D stands for *describe*.) This option is very helpful when you are writing EMACS extensions. If you are writing an extension and want to know the available functions that pertain to what you are doing, you can probably find them with `[Esc] [X] describe`.

When you type this command or `[Ctrl_] [D]` to access the Describe option, the following prompt is displayed in the echo area:

```
Describe (? for help) (q for exit):
```

Respond to the prompt by typing either the name of the function you want described or ? for help about the option. You can exit from the Describe option at this point by typing `q` or pressing `[Return]`.

## Specifying Function Names for the Describe Option

If you type an unprefixed character string, EMACS switches you to a special buffer and displays a list of all the EMACS functions whose names begin with the string you specified. The Describe option also gives information about the `&` character and other single-character nonalphanumeric printing characters.

If you type a character string preceded by `@`, EMACS displays a list (with descriptions) of all functions that contain the string within their function names.

If you type a character string preceded by `@@`, EMACS displays a list (with descriptions) of all functions that contain the string within their descriptions.

For example, if you type `search`, EMACS displays a list with descriptions of all functions whose names begin with `search`, such as `search_fd`. If you type `@search`, the list contains all functions having `search` within their names, such as `reverse_search`. Finally, if you type `@@search`, the list contains functions that have `search` within their definitions.

## Command Options

The list of functions in the special Describe buffer is often too long to be viewed on one screen. For this reason, the following viewer commands are available while you view the text in the Describe buffer:

<i>Option Name</i>	<i>Description</i>
<b>Space or Return</b>	Moves the cursor forward in the buffer to the next screen.
<b>b</b>	Moves the cursor backward in the buffer to the previous screen.
<b>l</b>	Refreshes the screen.
<b>p</b>	Prints a copy of the Describe buffer with the filename, <code>.viewer.print</code> .
<b>q</b>	Exits the Describe buffer and sends you back to your original text.
<b>r</b>	Reverse searches for a string.
<b>s</b>	Searches for a string.
<b>&gt;</b>	Moves the cursor to the end of the text in the buffer.
<b>&lt;</b>	Moves the cursor to the beginning of the text in the buffer.
<b>?</b>	Lists the above Describe viewer commands.

If you type a command other than those listed above, EMACS responds with an error message.

## Minibuffer Information

While you are viewing text in the Describe buffer, you will see the following message in the minibuffer area:

```
-- 15% -- Viewer (type ? for help)
```

The number tells you approximately what percentage of the text in the buffer is currently above the cursor. This can be very helpful when EMACS displays a long list of functions and you want to know how much information follows what is currently on your screen.

## L Option

The L help option displays a list in the minibuffer showing the last 20 characters you typed. (L stands for *list*.) It is helpful when you make a typing error that modifies your text and you do not remember what the error was. EMACS uses the caret (^) as a symbol for the Control key and `esc` to signify the Escape key when it lists the characters.

## ? Option

The ? help option lists all the above options for the `Ctrl _` command, as well as `Ctrl G` (`abort_command`).

---

# 5

## Speed-type

### Introduction

Speed-type is EMACS' abbreviation facility. When the speed-type environment is enabled, you can define an abbreviation for a single word or a large amount of text. Then, whenever you type the abbreviation during an EMACS session, EMACS expands it for you automatically. Three examples of speed-type use are:

- Abbreviation of several words with a single short word.
- Correction of misspelled or mistyped words.
- Changing words from lowercase to uppercase or vice versa.

This chapter explains how speed-type works. The first part of the chapter shows how to create speed-type abbreviations, how to save the abbreviations in a new speed-type file, how to use these speed-type abbreviations from within EMACS or by using a command line abbreviation, and how to add or delete abbreviations. The second part of the chapter is a reference section with detailed explanations of each speed-type command.

### How Speed-type Works

A speed-type abbreviation is a character string that you have defined during the current editing session or saved previously in a special abbreviations file. Each time you type a **separator** (such as a space) in the current buffer, EMACS checks to see if the text before point is an abbreviation. If it is, EMACS expands the abbreviation for you. For example, you could set up the abbreviation `ema` to stand for *EMACS minibuffer area*. Thereafter, each time you typed `ema` followed by a separator, EMACS would replace it with the words **EMACS minibuffer area**.

#### Note

EMACS expands the string even if you enter it in a command line within EMACS, so be careful not to define a component of a command, (for example, *add*), as an abbreviation.

The definition of a separator varies according to your current mode. In fundamental mode, any characters except letters or numbers are separators. `Return` and `Ctrl I`, which is the `type_tab` command, also act as separators in fundamental mode.

Language modes have their own separators. Speed-type keeps a list of the separators used in each mode. You can get this list with the `spd_list_all` command, which is described later on in this chapter.

EMACS commands (excluding `Return` and `Ctrl I`) are not considered separators. This means you can type an abbreviation, move to another place in your file, and then go back to the abbreviation and type the separator to cause expansion.

Before you can expand speed-type abbreviations, you must turn on the speed-type system. If you have already created a file of abbreviations that you want to use, you need to load it into EMACS, as described below.

## Creating Speed-type Abbreviations Interactively

To create a speed-type abbreviation, enter EMACS, turn on speed-type with the `spd_on` command, and create the abbreviation using the `spd_add` command. When the minibuffer displays the **Speed-Type symbol:** prompt, type your abbreviation. Then, when the minibuffer displays the **Expansion:** prompt, type the expansion of the abbreviation.

If the expansion is longer than the display line, you can type `Ctrl Q` (`^q_quote_command`) followed by `Return` to see text that extends beyond the right margin. The `Ctrl Q` command tells EMACS to interpret the `Return` as a normal carriage return instead of a terminator for your definition; therefore, the cursor moves to the second line of the minibuffer where you can continue typing your expansion. Terminate the expansion string by pressing `Return`.

If you want to create more than one speed-type abbreviation, go through the process again by typing `Esc X spd_add` and answering the minibuffer prompts.

### Note

To enter any speed-type command, type `Esc X` to cause the **Command:** prompt to appear in the minibuffer area of your screen. Then enter the speed-type command.

An example of creating an abbreviation is shown below. The minibuffer prompts are followed by your entries. (Press `Return` after each entry.)

Command: *spd\_on*

Command: *spd\_add*

Speed-Type symbol: *ny*

Expansion: *New York*

## Creating Template Abbreviations Interactively

A *template* abbreviation system is also available interactively. You can create expansions called *placeholders* that are simply sequences of characters that disappear as soon as you type text over them. The sequence is bounded by the symbols < and >.

To create a template, turn on speed-type and create a speed-type symbol. Type a placeholder in response to the **Expansion:** prompt. Respond yes or y to the next prompt, which asks if the expansion is a template. The next prompt declares fundamental mode and the placeholder. Respond yes to this prompt also.

An example of this procedure is shown below:

Command: **spd\_on**

Command: **spd\_add**

Speed-Type symbol: **bo**

Expansion: **<city>**

Is this a template: **yes**

Declare fundamental "<city>": **yes**

This creates a template called *bo*. When you type the characters **bo**, the placeholder **<city>** appears with the cursor resting on the < at the beginning of the word. The next character you type erases the entire placeholder.

If you answer no to the **Is this a template:** prompt, the characters **<city>** that you typed as your expansion appear when you type the abbreviation **bo**. But, in this case **<city>** is not a placeholder and the cursor does not rest on the < character. Rather, it rests after the space that follows the expansion.

More information about templates appears later in this chapter in the section called Adding Template Abbreviations to the Source File.

## Other Expansion Possibilities

There are other useful commands for creating abbreviations. Using **Ctrl X** **+** (**spd\_add\_region**) you can define a region as the expansion of the abbreviation, or you can create an abbreviation for the previous *n* number of words that you have typed. **Spd\_add\_modal** allows you to define an abbreviation for an EMACS mode. Refer to the Speed-type Commands Reference Section in this chapter for explanations of these commands.

## Saving Speed-type Abbreviations

Unless you save the abbreviations that you have just created, you will not be able to use them again after you terminate the current EMACS session. There are two ways to save your current abbreviations so that you can use them in a future EMACS session without redefining them. You need to save them in a special file, either during the current EMACS session or when you exit from it. Each method is discussed below.

To see a list of all your current abbreviations, type:

Command: ***spd\_list\_all***

### Note

These instructions do not pertain to a session when you have loaded a preexisting source file containing speed-type abbreviations into EMACS.

**Saving Abbreviations During an EMACS Session:** To save your abbreviations during your current session, use the `spd_save_file` command. EMACS prompts you for the name of your new speed-type file and creates a file with the suffix `.ESPD`. The following example shows the command, the prompts that appear in the minibuffer, and your responses.

Command: ***spd\_save\_file***

Create new Speed-type file?: ***yes***

Name of file: ***spabbrev***

Add "ny": ***yes***

Created SPABBREV.ESPD

### Note

If you name an existing `.ESPD` filename for the new file, EMACS issues another prompt:

Add "abbrev":

where "abbrev" is one of your new abbreviations. If your response is `y` or `yes`, EMACS overwrites the existing file with the current abbreviation(s).

**Saving Abbreviations in a File as You Exit from EMACS:** If you decide to save your abbreviations in a file just as you are leaving EMACS, with the `Ctrl X Ctrl C` command, answer `yes` or `y` to the **Save Speed-Type changes?**: prompt that appears in the minibuffer when you give the exit command. The procedure for saving the abbreviations, with your responses to the prompts, is as follows.

Save Speed-Type changes?: **yes**

Create new Speed-type file?: **yes**

Name of file: **spabbrev**

Add "ny": **yes**

Created SPABBREV.ESPD

If you have created more than one abbreviation, you may select which ones to save. Your prompts could look like this:

Save Speed-Type changes?: **yes**

Prompt for each change?: **yes**

Create new Speed-type file?: **yes**

Name of file: **spabbrev**

Add "ny": **yes**

Add "ny1": **no**

Add "ny2": **yes**

Created SPABBREV.ESPD

You now have the file SPABBREV.ESPD in your directory with two speed-type abbreviations in it. (If SPABBREV.ESPD already existed, EMACS overwrote it with the new abbreviations.) The next section explains how to activate this file when you enter EMACS.

## Using Saved Speed-type Abbreviations

There are two ways to load your abbreviation file into EMACS so that you can use the abbreviations:

- Issue a command from EMACS during your current session.
- Specify the command line option `-SPDT` when you start EMACS at PRIMOS command level.

This section discusses each method. Remember that to use the abbreviations, you must turn on the speed-type environment.

**Loading the Abbreviation File from EMACS:** You load an abbreviation file into the current EMACS environment with the `spd_load_file` command. The following example shows the command and the prompts that appear in the minibuffer.

Command: ***spd\_on***

Command: ***spd\_load\_file***

File to load: ***spabbrev***

Loaded SPABBREV.ESPD

Note that EMACS has added the `.ESPD` suffix to your abbreviation file. When EMACS creates an abbreviations file, it automatically assigns it the suffix `.ESPD`. Similarly, EMACS appends an `.ESPD` suffix to the filename you specify to load. You do not add the `.ESPD` suffix when you specify the file.

If you name a file that does not exist, the following message appears:

Loading Speed-Type FILENAME.ESPD: Not found.

**Loading the Abbreviation File and Turning on Speed-type with Command Line Option `-SPDT`:** The second method of using speed-type is to use a PRIMOS command line option that both turns on speed-type and loads a speed-type abbreviation file automatically when it starts an EMACS session. The format of the option is as follows:

**`-SPDT` *pathname***

where *pathname* specifies a speed-type abbreviations file.

If your speed-type abbreviations are in a file called `SPABBREV.ESPD`, your EMACS command line could look like this:

```
EMACS [pathname] [arguments] -SPDT <DISKNAM>UFD>SPABBREV
```

Note that you do not type the `.ESPD` suffix when you type the *pathname* of `SPABBREV`. The position of the `-SPDT` option on the command line is optional.

#### Note

EMACS command line arguments are described in Chapter 1 of this book. Chapter 1 also explains how to construct a PRIMOS abbreviation for your command line using the `-SPDT` option so that speed-type is turned on each time you start EMACS.

## Adding and Deleting Abbreviations Interactively

You may add abbreviations to your current speed-type environment at any time by turning on speed-type, if it is not already on, and typing the `spd_add` command. To delete a single abbreviation, use the `spd_delete` command. An example of the command and prompt for deleting the abbreviation `ny` is shown below:

Command: **`spd_delete`**

Speed-Type symbol to delete: **`ny`**

To deactivate all of your abbreviations temporarily, use the `spd_off` command. Issuing the `spd_on` command reactivates the abbreviations.

If you have loaded a speed-type abbreviation file into EMACS and want to add and save more abbreviations, follow the procedures in the sections `Creating Speed-type Abbreviations` and `Saving Speed-type Abbreviations`.

If you add one new abbreviation and do not save it in your current session, the following message appears as you leave EMACS:

Save Speed-Type changes?:

If you answer yes, EMACS gives you a brief message that it rewrote your abbreviation file and exits. If you answer no, there is no message.

If you have added more than one abbreviation to your abbreviation file that you have loaded into EMACS, and exit EMACS without saving them, the prompts with your responses would look like this:

Save Speed-Type changes?: **`yes`**

Prompt for each change?: **`yes`**

Add "ny": **`yes`**

Add "ny1": **`no`**

Add "ny2": **`yes`**

Rewrote SPABBREV.ESPD

## Editing the Abbreviation Source Files

It is possible to add or delete abbreviations by editing the abbreviation file, rather than by entering or deleting them interactively. Speed-type provides a way to do this from within a special EMACS buffer called `.spd_list`.

When you request a list of current abbreviations with either the `spd_list_all` or `spd_list_file` command, EMACS creates the list based on the abbreviation file(s) that have been loaded into the current speed-type environment. Because this list is placed in the `.spd_list` buffer, you can edit it just like an ordinary EMACS file. If no abbreviation file has been loaded, the `spd_list_all` command displays the empty `.spd_list` buffer, in which you can create a source abbreviation file.

#### Note

If you change the abbreviation file, you have to compile it and load it again before you can use the added or changed abbreviations.

To enter the `.spd_list` buffer, give one of the following commands:

Command: `spd_list_all`

or

Command: `spd_list_spabbrev`

If you have loaded the SPABBREV abbreviation file that contains one abbreviation called `ny` into the environment, the `.spd_list` buffer display appears as shown below with the cursor resting on the home position of your screen:

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
mode fundamental "^i^j !~"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~■"
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

** ny **

"New York"
=====

```

The status line declares that the current mode is fundamental and lists the separators for fundamental mode.

When you add or delete abbreviations to or from the source file, follow these formatting rules:

- Begin the line containing the name of the abbreviation in column 1 with two asterisks (\*\*), followed by a space. The name must be followed by a space and two asterisks. In the example above, the name of the abbreviation is `ny`.
- Separate abbreviations by a line having at least one equal sign (=), which is in column 1. An entire line of equal signs is recommended.
- Begin and end each abbreviation's expansion with a double quote (").

Blank lines and comment lines are optional in the source file. They exist only to make the listing more readable. You can type blank lines and comment lines (those beginning with semicolons) before and after an abbreviation and its expansion, and before and after lines containing equal signs. The only place a blank line is not ignored is within an abbreviation's expansion. If you include a blank line within a set of double quotes, it becomes part of the expansion.

For an example, suppose you were adding two abbreviations to a source file listing. You might type the following:

```
=====
; abbreviation added on 2/14/87
** ORGR **
"ORGANIZATION IS RELATIVE; ACCESS MODE IS RANDOM,
RELATIVE KEY IS"
=====
** USEA **
"USE AFTER STANDARD EXCEPTION PROCEDURE ON INPUT."
=====
```

## Adding Template Abbreviations to the Source File

You can also add templates to the source file in the `.spd_list` buffer by editing it. Templates allow you to create an expansion containing placeholders, which are inserted in your file during your EMACS editing session.

Placeholders are simply sequences of characters that disappear as soon as you type text over them. They are bounded by the symbols `<` and `>` in an abbreviation source file.

When you add new templates to the source file, follow these formatting rules:

1. Declare each placeholder in the mode declaration section of the source file so that EMACS can distinguish it from ordinary text. You can list the placeholders in any order. They do not have to correspond to the order in which they appear in the template.
2. Type **template** after the second set of asterisks (**\*\***) on the abbreviation definition line.
3. Type the text of the template, beginning and ending with a double quote (**"**). Include the placeholder, bounded by `<` and `>`, in the text.

Two examples of templates are shown below. The first example shows templates that can be used to facilitate document formatting with RUNOFF. Each template contains an example of RUNOFF code for a format element. Included in this example are templates for a bullet list, a note, and first-, second-, and third-level headings. The first line of each template is a comment that describes its contents. Note that the RUNOFF code is enclosed in double quotes.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
mode fundamental "^i^j !~\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~■"
;
place_holder <Text of bullet>
place_holder <Last item>
place_holder <Text>
place_holder <FIRST-LEVEL HEADING>
place_holder <Second-level Heading>
place_holder <Third-level Heading>
place_holder <Number>
place_holder <Title>
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

=====
;Template for a bullet list.

** bl ** template

.in7
.p -3 1
o``<Text of bullet>
.p
o``<Last item>
.un7
.sk"
=====
;Template for a note.

** nt ** template

.sk2
.>{{Note}}
.sk
.ri4
.in4
<Text>
.run4
.un4
.sk2"
=====

```

```
;Template for a first-level heading.
```

```
** flh ** template
```

```
".sk 3
```

```
<FIRST-LEVEL HEADING>
```

```
.sk"
```

```
=====
;Template for a second-level heading.
```

```
** slh ** template
```

```
".sk3
```

```
<Second-level Heading>
```

```
.sk"
```

```
=====
;Template for a third-level heading.
```

```
** tlh ** template
```

```
".sk 2
```

```
<Third-level Heading>:"
```

In the sample listing, *bl*, the first abbreviation, is a template containing two placeholders. The second abbreviation, *nt*, is a template containing one placeholder. The three abbreviations for headings each contain one placeholder.

The second example of a speed-type template shows how to use templates to implement a language format (COBOL, in this case):

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
mode cobol ";^j !~"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~■"
```

```
;
```

```
place_holder <boolean>
```

```
place_holder <statements>
```

```
place_holder <number>
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
=====
** IFE ** template
```

```
"IF <boolean>
```

```
  <statements>
```

```
  ELSE
```

```
    <statements>."
```

```
=====
```

```
** MIW **
```

```
"MOVE INPUT-STRUCTURE WS-STRUCTURE.  
ADD 1 TO COUNTER."
```

```
=====
```

```
** MIC **  template
```

```
"MOVE INPUT-STRUCTURE WS-STRUCTURE.  
ADD <number> TO COUNTER."
```

```
=====
```

Note that in both examples, the placeholders are declared in the mode declaration section of the file in addition to being defined in the abbreviation. You can also delete placeholder abbreviations from the abbreviation source file. Remember to maintain the proper format of the remaining sections.

## Template Expansion

After expanding a template abbreviation in EMACS text, EMACS leaves the cursor at the first placeholder in the expansion. Since it assumes you want to fill in the placeholder, the next character you type erases the entire placeholder. For example, if you had typed the abbreviation IFE from the sample source listing above, EMACS would insert the text below into your file at point. The position of point after expansion is indicated by an underscore in this example:

```
IF <boolean>  
  <statements>  
ELSE  
  <statements>.
```

If there are no placeholders in a template, EMACS places point after the expansion, as usual.

If you do not remove a placeholder from your text, it becomes part of the text. If you are entering a program, the language compiler may flag it as an error.

EMACS has two special commands for moving to placeholders.

►  Ctrl X  >

**forward\_place\_holder**

This command moves point to the next placeholder without erasing or altering the current placeholder. Point is left at the leading < character of the next placeholder.

### Note

If you bind  Ctrl X  > to a printing key, execution of the command erases the current placeholder before moving point to the next one. It is more efficient to use  Ctrl X  > to move to the next placeholder, making it possible to locate the current placeholder later in your editing session if you need to return to it.

▶ `Ctrl X` `<`**back\_place\_holder**

This command moves point to the previous placeholder. Point is left at the leading `<` of the placeholder.

## Changing Mode Separators in a Source File

By editing a source file, you can change the list of separators for a mode. (See the section, *How Speed-type Works*, above, for information about separators.) To change the list, add or delete characters in the first line of the mode section.

If you are creating your own mode, add the list of separators after the mode name. If you do not add a mode separator list, EMACS will use the separators for fundamental mode.

## Compiling and Loading the Source File

After you have finished making changes to the source file, you can compile the file, using the `spd_compile` command. In the following example, the name for the output file is RUNO.

Command: **`spd_compile`**

Name of output file: **`runo`**

Compilation completed

EMACS adds the suffix `.ESPD` to the name of the output file.

To load the new file, use the `spd_load_file` command. The following example shows the command and the responses for loading RUNO.

Command: **`spd_load_file`**

File to load: **`runo`** or **`*>mydir>runo`**

Loaded RUNO.ESPD

The abbreviations in your source file are now ready for expansion in your current EMACS session.

## Speed-type Commands Reference Section

This section discusses how all the EMACS speed-type commands work. For information about adding a speed-type command to an EMACS library file, or binding a speed-type command to a key, see the *EMACS Extension Writing Guide*.

**▶ back\_place\_holder**

Ctrl X &lt;

**Usage:** Moves point to the previous placeholder.

Point is left at the leading &lt; character of the placeholder.

**▶ forward\_place\_holder**

Ctrl X &gt;

**Usage:** Moves point to the next placeholder.

Point is left at the leading &lt; character of the placeholder.

**▶ spd\_add****Usage:** Adds an abbreviation to the current speed-type environment.

This command does *not* save the abbreviation permanently. When you type this command, EMACS prompts you for the name of the abbreviation you want to add and the expansion of the abbreviation.

If the expansion is longer than the display line, you can type `Ctrl Q` (`^q_quote_command`) to see text that would otherwise extend beyond the right margin. The `Ctrl Q` command tells EMACS to insert the `Return` as a new line character in the expansion, instead of using it to terminate your definition.

**▶ spd\_add\_modal****Usage:** Defines an abbreviation specifically for a mode, such as fundamental, overlay, or fill mode, or for programming language modes.

After prompting you for the abbreviation and its expansion, EMACS prompts you for the name of the mode in which you want the abbreviation to be active. If the mode you specified is not turned on at the time of definition, the abbreviation will not work. It is enabled only when the mode is turned on.

When EMACS expands a modal abbreviation, it follows the rules of that mode. In COBOL mode, for example, expanded text is placed in the appropriate columns automatically.

An example of this command is shown below:

Command: ***spd\_add\_modal***Speed-Type symbol: ***ls***Expansion: ***Los Angeles***Mode: ***fill***

  
▶ `spd_add_region``Ctrl X` `+`

**Usage:** Defines a region in text as the expansion of an abbreviation.

When you type `Ctrl X` `+`, EMACS inserts the word **speed\_type** into the status line. If you move the cursor and type `Ctrl X` `+` again, EMACS considers the intervening text to be a region, and prompts you for the abbreviation. After you enter the abbreviation, EMACS moves the cursor back to its original position. If there are any leading or trailing spaces within the region, EMACS removes them from the definition of the abbreviation.

You can give `Ctrl X` `+` an argument and it will take the specified number of words preceding the cursor as the expansion for an abbreviation. For example, if you type `Esc` `3` `Ctrl X` `+`, EMACS takes the three words preceding the cursor as the expansion and prompts you for the abbreviation.

  
▶ `spd_compile pathname`

**Usage:** Compiles the speed-type source file in the current buffer and writes it to the binary output file specified by *pathname*.

If you do not specify a pathname, EMACS prompts you for it. EMACS will add the suffix `.ESPD` to whatever filename you specify.



To use the newly compiled abbreviations in the current EMACS session, you must load the new file using the `spd_load_file` command.

  
▶ `spd_delete`

**Usage:** Deletes an existing abbreviation from the speed-type environment.



When you type this command, EMACS prompts you for the name of the abbreviation you want removed. After you respond, it removes the abbreviation from the current speed-type environment. If the abbreviation occurs in different modes, EMACS removes it only from the current active mode.

When you exit from the current EMACS session, EMACS gives you the **Save Speed-Type changes?:** prompt. If you answer yes, and you loaded an `.ESPD` file during the current session, EMACS overwrites the `.ESPD` file (or the first file read in, if more than one). If you answer no, EMACS ends the session without saving the changes.

  
▶ `spd_list`

**Usage:** Prints information for you about an abbreviation that has been loaded into your speed-type environment during the current EMACS session.



When you type this command, EMACS prompts for the name of the abbreviation. After you respond, EMACS prints the text of the abbreviation (and other information associated with the it) at the top of the screen. The information display disappears when you type a new character or command.

► `spd_list_all`

**Usage:** Gives you information about all abbreviations currently loaded into the speed-type environment.

When you type this command, EMACS switches you to a special buffer called `.spd_list`, which contains a list of current abbreviations and their expansions. Here is a sample listing:

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
mode fundamental "^]i^j !~"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~■"
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

** ema **

"EMACS minibuffer area"
=====
** bos **

"Boston "
=====

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
mode fill "^i^j !~"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~■"
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

** aqr **

"Albuquerque"
=====
** sf **

"Santa Fe"
=====

```

Abbreviations in this listing are grouped according to mode rather than according to file. Fundamental mode always appears first, and other modes follow in alphabetical order. Semicolons distinguish one mode listing from another. Abbreviations are separated from one another by lines of horizontal equal signs.

In the sample listing above, the only modes are fundamental mode and fill mode. The symbols following the mode notation are the separators used in the mode. The first abbreviation listed is *ema*. The following line of text, **EMACS minibuffer area**, is the expansion for the abbreviation. Expansions always begin and end with double quotation marks (").

► `spd_list_file pathname`

**Usage:** Lists abbreviations found in the file specified by *pathname*.

This command is similar to `spd_list_all` described above, except that it lists only the abbreviations found in the file specified by *pathname*.

If *pathname* is not given, EMACS prompts you for the name of the speed-type file. If you enter the name of a file that you have not loaded into your current environment, an error message notifies you that the file is not in the database.

► `spd_load_file`

**Usage:** Loads an abbreviation file into the speed-type environment.

When you type this command, EMACS prompts you for a filename. (Do not append the suffix `.ESPD` when you enter the filename.) You can load up to ten files during one editing session. If more than one file contains an abbreviation with the same name, EMACS expands the abbreviation as it is found in the most recently loaded file.

► `spd_on`

**Usage:** Turns on the speed-type system.

Abbreviation expansion occurs if an abbreviation file has been loaded into EMACS or if you have created abbreviations during the current session.

► `spd_off`

**Usage:** Turns off the speed-type system.

Any abbreviations files you have loaded remain available for use later in the editing session in case you turn on speed-type again.

After you type this command, you can still add, delete, and display abbreviations, but EMACS will not expand them.

► `spd_save_file`

**Usage:** Saves all the changes, including deletions, you have made to the current speed-type environment.

It overwrites the *first* speed-type file you loaded into EMACS. (This file is referred to as the *primary* file.)

▶ **spd\_unexpand**

Ctrl X -

**Usage:** Removes expansion from current buffer.

This command removes the most recently expanded text from the current buffer and reinserts its abbreviation back into the text.

The cursor can be anywhere in the current buffer when you type this command.

---

# 6

## Customized Library Files

### Introduction

An EMACS library file typically includes commands and functions that set up your individualized EMACS environment, commands that turn on certain EMACS modes such as fill mode, or commands that set up customized keybindings.

This chapter provides the following:

- A discussion about how to create, save, and execute a library file, as well as how to make library file commands available automatically at the beginning of each EMACS session.
- A section on the file hooks mechanism.
- Instructions for adding keybindings to your library file.

### User Library Files

A user library file is a standard PRIMOS ASCII text file that contains Prime EMACS Extension Language (PEEL) statements. For complete instructions about writing PEEL statements, see the *EMACS Extension Writing Guide*.

An example of a user library file is shown below:

```
(go_to_buffer "main")
(fill_on)
(setq default_right_margin 60)
(set_permanent_key "~c[OL" "delete_char")
(setq user_type$ 'clerical$)
```

The commands in this example turn on fill mode in the buffer called main, set the right margin to column 60, bind the PF12 key (on the PT200 terminal) to the `delete_char` function, and set your user type to `'clerical$`.

## How to Create, Save, and Execute a Library File

Your library file should include the commands that are necessary to set up your EMACS working environment. An example of this type of command appears in the user library file example above. The `setq` function, applied to the `user_type$` variable, sets your user type so that certain modes are in effect when EMACS reads in a file. (User types are explained in the section on file hooks, later in this chapter.)

The library file may also contain commands which facilitate and simplify your particular task of editing or text entry. An example of this type of command is the `set_permanent_key` command that binds the PF12 key on the PT200 terminal to the `delete_char` function. By binding your customized functions to function keys on your terminal, you save keystrokes.

You can assign values to variables in a library file. For example, fill mode has a default right margin of 70. If you want your right margin always to have a value of 60, rather than 70, you may add the following statement to your startup library file (as shown above):

```
(setq default_right_margin 60)
```

After you have created the source code, you can make your file into an EMACS library file using one of two methods:

### Method One

1. Save the source code in a file with an `.EM` suffix, for example, `USER_LIB.EM`.
2. Compile, load, and execute the `.EM` file at the beginning of each session with the following command. Use `[Esc] [X]` to get to command mode.

Command: **`load_pl_source`**

EMACS prompts you for the pathname of the source file.

3. Respond with the name of the `.EM` file (for example, `USER_LIB.EM`). Then, EMACS finds, compiles, and loads the file and executes the functions. Note that `load_pl_source` does not execute commands in your file that are used with `[Esc] [X]`, but makes them available for execution.

### Method Two

1. From the buffer containing the source code file, compile and save the file with the following command:

Command: **`dump_file`**

This causes a `fasdump` operation. EMACS stores the compiled file in a file with the suffix `.EFASL`.

2. From the same buffer, issue the following command to load the compiled file:

Command: **`load_compiled`**

3. Respond to the **Fasdump file name:** prompt as follows. Note that you do not have to type the .EFASL suffix here.

Fasdump file name: **user\_lib**

The file USER\_LIB.EFASL is now in the directory you specified and is ready for EMACS to call at startup time.

## Loading a Library File Automatically

By using the `-ULIB` option in your EMACS command line, you can indicate that you want the commands in a specified user library file to execute automatically when you invoke EMACS. The following example shows a command line that calls a library file:

OK, **EMACS MY\_FILE -TTP PT200 -ULIB USER>STARTUP.EM**

This command line causes EMACS to load the file called MY\_FILE into an EMACS buffer, to recognize your terminal as a PT200, and to compile, load, and execute the commands contained in the library file whose pathname is USER>STARTUP.EM. If you do not include a suffix on the command line, EMACS looks for a library file with an .EM suffix first. If there is no file with an .EM suffix, it looks for a file with an .EFASL suffix.

If you create an abbreviation for your EMACS invocation command line that contains the `-ULIB` option, the specified library files are loaded automatically whenever you use the abbreviation to start an EMACS session. A .EFASL file loads faster than its corresponding .EM file. Remember to recompile the .EFASL file if you modify its counterpart .EM file.

## File Hooks

EMACS is used by many people who have different needs. For example, when you are typing a report, you usually want fill mode on so that you do not have to type carriage returns. However, fill mode is not really useful when you are producing COBOL programs. Some people like EMACS to be in overlay mode, but others use overlay mode only while editing tables. Consequently, EMACS has a mechanism, called *file hooks*, that allows you to specify what EMACS should do after it reads in a file.

When EMACS finds a file, it can check the file suffix, using the `found_file_hook` function, to determine which mode to use. The file hooks mechanism can also check to see what category of user is using EMACS. EMACS supplies three file hook categories.

<i>Category</i>	<i>Description</i>
clerkal\$	Invokes fill mode, unless you enter EMACS without specifying a file-name.
programmer\$	Always invokes default modes for file type.
no_file_hooks\$	No action is performed.

You or your System Administrator may create additional user types.

### Note

The file hooks mechanism for `clerkal$` does not place the EMACS buffers called *main* and *alternate* in fill mode. If you want these buffers always to be in fill mode, place the following commands in your startup library file.

```
(go_to_buffer "main")
(fill_on)
(go_to_buffer "alternate")
(fill_on)
```

If these commands are in your library file, the buffers are placed into fill mode before EMACS reads in a file.

For the `programmer$` user type, EMACS performs the following actions, depending on the suffix of the file you select to edit.

<i>Suffix</i>	<i>EMACS Action</i>
<b>RUNI</b>	Invokes fill mode
<b>EM</b>	Invokes LISP mode
<b>LISP</b>	Invokes Common LISP mode
<b>CBL</b>	Invokes COBOL mode
<b>FTN, F77</b>	Invokes FORTRAN mode
<b>RPG</b>	Invokes RPG mode
<b>VRPG</b>	Invokes RPG mode
<b>C or CC</b>	Invokes C mode
<b>COMO</b>	No action taken

## Setting a File Hook

To set a file hook, use the `setq` function to initialize the `user_type$` variable. For example, the following statement causes EMACS to use the `'programmer$` file hook:

```
(setq user_type$ 'programmer$)
```

If you create a user type called `data_entry$`, you would use the following statement:

```
(setq user_type$ 'data_entry$)
```

To set your user type in a library, include one of the following statements:

```
(setq user_type$ 'programmer$)
(setq user_type$ 'clerkal$)
(setq user_type$ 'no_file_hooks$)
(setq user_type$ 'user_defined$)
```

where `'user_defined$` is a type that has been created and defined according to the procedure described below. The `setq user_type$` command should be the last command in your library.

## Creating and Changing File Hooks

You can create your own user type by writing a PEEL function. Refer to the *EMACS Extension Writing Guide* for information about writing functions. The following example shows the structure of a function for a programmer\$ user type. The semicolons are followed by explanatory comments.

```
(defun programmer$ ()
  (select (suffix$)
    "runi"
      (fill_on)          ; text mode
    "em"
      (lisp_on)          ; lisp mode
    "lisp"
      (cl_on)           ; common lisp mode
    "cbl"
      (cbl_on)          ; cobol mode
    "ftn" "f77"
      (fortran_on)      ; fortran mode
    "vrpg"
      (vrpg_on)         ; rpg mode
    "c" "cc"
      (cc_on)           ; c mode
    "como"
      ()
    otherwise
      ()
  ))
```

This function consists of three parts:

1. The defun (define function) statement, followed by the name of the user type.
2. The select statement, followed by the word *suffix\$* within parentheses. The select statement evaluates and compares the values to determine the appropriate action. The *suffix\$* function checks a file's suffix.
3. A list of suffixes and what actions to perform.

Suppose you have a file type that has a suffix of *rpt* (for report) which consists mainly of tables. When you are editing reports, you do not want to change the column positions. You could add the following line to the list of suffixes and actions:

```
"rpt"
  (overlay_on)          ; overlay mode for tables/reports
```

Suppose the report document always contains a five-line header that you want to skip. You could add the following code:

```
"rpt"  
  (overlay_on)  
  (if (first_line_p)  
      (next_line_command 5))
```

where **if** and **first\_line\_p** are PEEL statements. Here they ask: "Is point on the first line of the buffer?" Note that the argument to the **next\_line\_command** command must be contained within the parentheses.

If you know the keypath, but not the name of a command you would like to include in the code, type `Ctrl_` and respond to its query with C. When you type in the actual keypath, EMACS displays the name of the command and a brief description of it.

After you have created a function, the name of the function becomes a new user type. For example, if you create a function called `data_entry$`, then `data_entry$` becomes the user type:

```
(defun data_entry$  
  (select (suffix$)  
    ...  
  ))  
  
(setq user_type$ 'data_entry$)
```

## Creating Your Own Interface

EMACS can make use of your terminal's function keys that are situated across the top and on the right and left sides of the keyboard. Use the `set_permanent_key` command to bind a customized function to one of the terminal keys. This function has the following structure:

```
(set_permanent_key "keypath" "command")
```

For example, on the PT200, function key PF12 transmits the following sequence:

```
[Esc] [O] [Esc] [L]
```

This sequence is represented by the character sequence `"~c[OL"`. If you want to bind the `delete_char` function to the PF12 key, simply insert the following function in your source file:

```
(set_permanent_key "~c[OL" "delete_char"))
```

(The conventions for typing keystrokes to this command are described in Chapter 2 of the *EMACS Extension Writing Guide*.)

The following files contain lists of function key keypaths for the PT45, the PST 100, and the PT200 terminals.

```
EMACS*>INFO>KEY_ASSIGNMENTS.EM
EMACS*>EXTENSIONS>SOURCES>PT45_FUNCTION_KEYS.EM
EMACS*>EXTENSIONS>SOURCES>PST100_FUNCTION_KEYS.EM
EMACS*>EXTENSIONS>SOURCES>PT200_FUNCTION_KEYS.EM
```

If you are rebinding for your own use, the library file procedure meets your needs completely. You can bind EMACS functions to any keystrokes. In addition, you can create your own functions and bind them to keys.

However, to set up an interface that many people will use, you will want to do things in a more general way, as described by the following steps:

1. Create a file that contains all the function key definitions. For example,

```
(setq pf12_pt200$ "~c[OL")
```

2. Create a file that contains all the commands that will exist in your command set. Adding new commands is permitted.
3. Create a third file that contains the actual bindings. For example, if you defined in Step 1 what function key PF12 is, you might type

```
(set_permanent_key pf12_pt200$ "delete_char")
```

You would have one of these statements for every new assignment.

4. Modify the startup library file so that the files are loaded, and these bindings will come up automatically. Your entries in the library file could be:

```
(fasload "key_assignments")
(fasload "extension_file")
(fasload "binding_file")
```

Creating separate files lets you use different bindings or definitions for different circumstances. For example, suppose that you want to create two sets of bindings for one terminal. By placing the key assignments separate from the definition of the bindings, you can use the definitions in more than one way. Also, this system allows you to bind equivalent functionality to more than one kind of terminal even if the terminals do not have ASCII standard function keys.

## Simplifications You Can Make

### Toggles

In a keybinding set, it is often desirable to have one key that performs more than one function. For example, suppose you want to use function key PF12 so that the first time it is used, it turns on overlay mode, and the second time, it shuts off overlay mode. This may be done as follows.

```
(defcom toggle_overlay_on
  (overlay_on)
  (set_permanent_key "~c[OL" "toggle_overlay_off"))

(defcom toggle_overlay_off
  (overlay_off)
  (set_permanent_key "~c[OL" "toggle_overlay_on"))

(set_permanent_key "~c[OL" "toggle_overlay_on"))
```

The sequence "**~c[OL**" is assumed to be the keystroke characters sent by the terminal. The first time PF12 is typed, it activates the **toggle\_overlay\_on** function. As it executes, it changes the definition of PF12 so that the next time it is typed, it activates the **toggle\_overlay\_off** function. When this function executes, it changes the binding on PF12 so that **toggle\_overlay\_on** is now bound to PF12.

#### Note

This toggling can be used with nearly all EMACS functions. The exceptions are **collect\_macro** and **finish\_macro**.

The toggle procedure has one inherent weakness. If the toggle sets a mode, the mode is buffer specific. However, when you switch to a different buffer, problems may occur because the buffer is not synchronized with what the function expects. One solution is to use a **set\_key** command rather than a **set\_permanent\_key**. This command localizes the change to your current buffer.

A more general solution is to determine what state the buffer is in, with the **buffer\_info** function. The **buffer\_info** function is used in the following example to check a buffer's mode list. Depending on what it finds, it turns overlay either on or off. For example,

```
(defcom toggle_overlay
  doc "Alternates between overlay mode on/off"
  (if (member (find_mode 'overlay) (buffer_info modes))
      ; when in here, overlay mode is ON already
      (overlay_off)
    else
      (overlay_on)))
```

You can simulate this procedure for functions that are not modes by using **buffer\_info**'s user list. (For more information on using the **buffer\_info** function, see the *EMACS Extension Writing Guide*.) The toggle method presented at the beginning of this section is the one to use when moving back and forth between procedures that are globally set.

## Menus

It is inefficient to create menus that let the user select only one option. Here is a small menu function that puts **help\_on\_tap** functions onto a menu.

```
(defcom help_menu
  doc "Presents help stuff on a key"
  (save_excursion
    (go_to_buffer ".help_menu")
    (if (empty_buffer_p)
        (insert "1    Apropos~n")
        (insert "2    Explain~n")
        (insert "3    Describe~n")
        (insert "4    Wallpaper~n")
        (buffer_info changed_ok true)
        (buffer_info read_only true)
        (buffer_info dont_show true))

    (select (prompt "Help")
            "1" (apropos)
            "2" (explain_key)
            "3" (describe)
            "4" (wallpaper)
            otherwise
              (ring_the_bell)
              (info_message "Unknown function"))))
```

For explanations of the functions in this menu, refer to the *EMACS Extension Writing Guide*.

After you have saved, compiled, and loaded this file, you can execute the menu function by typing `[Esc] [X] help_menu`. For explanations of the `help_on_tap` command, see Chapter 4, Online Help Commands.

---

# 7

## The TERMCAP Facility

### Introduction

TERMCAP is the name of a file that contains information about the operating features of many different terminals. The TERMCAP file is located in the directory EMACS\*>TERM. A TERMCAP description for any given terminal is called an *entry*. At present, the TERMCAP file contains entries for approximately two hundred terminals. Some of these entries contain *capabilities* that are not supported by Prime and are ignored by EMACS. This chapter describes only the TERMCAP capabilities that are currently supported by EMACS.

Terminals can differ greatly from one another. Before TERMCAP was developed, a programmer had to rewrite portions of the source code to create individual drivers for each terminal and then recompile the entire editor program. With TERMCAP, a programmer can add an entry to the TERMCAP file to support a new terminal without revising or recompiling the source code for the editor. Because the file is used whenever EMACS is invoked, all users on a system can use any entry in the TERMCAP file.

This chapter is divided into three parts:

- Using TERMCAP with EMACS
- TERMCAP capabilities
- Adding a new terminal description to the TERMCAP database

#### Note

The TERMCAP file is in the public domain and is made available to the user by courtesy of Prime Computer, Inc. Prime makes no representations or warranties whatsoever regarding this file, or the ability of any Prime software, when combined with this file, to operate on any terminals other than Prime terminals. Prime also disclaims any obligations to maintain or support this file or any similar file now or in the future.

## Using TERMCAP With EMACS

There are three ways to indicate your terminal type to EMACS. You can use the `-TTP` option on the command line that invokes EMACS:

**EMACS filename `-ttp terminal_type`**

EMACS also recognizes the global variable, `.TERMINAL_TYPE$`. If you use the `-TTP` option or if you set `.TERMINAL_TYPE$`, EMACS looks in the TERMCAP file for the definition of the specified entry. Another global variable, `.TERMCAP$`, lets you specify a pathname to a TERMCAP entry that is *not* in the standard TERMCAP file. Setting these two global variables is explained in the following section.

### Global Variables

You can set the two PRIMOS global variables:

**`.TERMINAL_TYPE$`** This global variable contains the name of the terminal you wish to use. It will be used if you do not specify the `-TTP` option on the command line.

**`.TERMCAP$`** This global variable contains the user's TERMCAP database pathname. If it is not set, the default is `EMACS*>TERM>TERMCAP`. By defining a terminal in a separate file, a user can debug an entry before adding it to the TERMCAP database. The `TERMCAP$` variable specifies the pathname of the file that contains the entry.

Before using either of these variables, you must create and activate a global variable file, which will contain the variable definition. (See the *Prime User's Guide* for more information about global variables.) To define `.TERMINAL_TYPE$` as `PETE`, for example, you would enter this line in the global variable file:

```
SET_VAR .TERMINAL_TYPE$ PETE
```

To define the variable `.TERMCAP$` as the pathname `<DSKNAM>MYDIR>TERMCAP>PST`, you would enter this line in the global variable file:

```
SET_VAR .TERMCAP$ <DSKNAM>MYDIR>TERMCAP>PST
```

Assume that `<DSKNAM>MYDIR>TERMCAP>PST` contains the definition of `PETE`. Now, if you invoke EMACS without using the `-TTP` option, the `.TERMINAL_TYPE$` variable definition, `PETE`, specifies the terminal type; the `.TERMCAP$` variable definition gives the pathname to `PETE`'s definition in the file `PST`.

If you do not give a terminal name, either in the command line or in a global variable, the following message appears:

```
OK, emacs
```

```
A terminal type has not been specified.  
You must specify a terminal type which EMACS recognizes.
```

```
EMACS recognizes the PT200, PST100, PT45 and all  
other terminal types defined in the TERMCAP data base  
file [pathname: EMACS*>TERM>TERMCAP].
```

```
Please enter your terminal type  
if it is among those which  
EMACS recognizes, or enter the word "none":
```

To find out which terminals are in the TERMCAP database, you can look at the TERMCAP database file in EMACS\*>TERM.

#### Note

The `-TTP` option used on the command line overrides the global variable containing the name of the desired terminal type. An example is shown below.

```
EMACS -TTP PST100 -NOXOFF -ULIB <DSKNAM>MYDIR>STARTUP.EFASL
```

By issuing this command, you would access the PST 100 rather than the PETE terminal.

When you specify a terminal with `-TTP` that is not the PST 100, PT200, or PT45, EMACS searches the TERMCAP database for an entry that describes that terminal. If you give an incorrect terminal name (not the PST 100, PT200, or PT45), for example, XYZ, the following message appears:

```
OK, emacs -ttp xyz
```

```
The terminal type specified is not contained in the TERMCAP  
data base file.
```

```
XYZ is not a terminal type which EMACS recognizes.  
You must specify a terminal type which EMACS recognizes.
```

```
EMACS recognizes the PT200, PST100, PT45 and all  
other terminal types defined in the TERMCAP data base  
file [pathname: EMACS*>TERM>TERMCAP;.
```

```
Please enter your terminal type  
if it is among those which  
EMACS recognizes, or enter the word "none":
```

## TERMCAP Capabilities

This section contains information that will help you prepare your own TERMCAP entry. The following topics are discussed:

- Data types of the TERMCAP capabilities
- TERMCAP capabilities grouped according to category (Table 7-1)
- Discussion of the capabilities
- Control characters, delay padding, and placeholders

### Data Types

There are three data types for TERMCAP capabilities:

- **Boolean:** A Boolean value is either on or off. When you include a capability in your entry that takes a Boolean value such as `in`, it means that the insert mode that distinguishes nulls is turned on.
- **Numeric:** Terminal characteristics such as the number of lines or columns are numeric. The format of a capability that takes a numeric data type is the name, followed by a sharp character (`#`), then the numeric value. For example, `li#24` shows the number of lines that a screen displays.
- **String:** Capabilities that take a string data type are those that perform a specific function. The format of a capability that takes a string data type is the name, followed by an equal sign (`=`), then the string. For example, `cl=^z` represents the clear screen feature.

The EMACS TERMCAP handler has five categories of capabilities:

- Basic terminal features
- Screen movement
- Screen update
- Software control of terminal
- Special characteristics

Table 7-1 lists the capabilities by category. Within each category the data type and feature of each capability are described. Following the table are descriptions of each category.

**Table 7-1**  
**TERMCAP Capabilities by Category**

<b>Basic Terminal Features</b>		
<i>Name</i>	<i>Data Type</i>	<i>Description</i>
co	Numeric	Number of columns on screen
li	Numeric	Number of lines on screen
<b>Screen Movement</b>		
<i>Name</i>	<i>Data Type</i>	<i>Description</i>
bc	String	Backspace character if not ^h
cm	String	Cursor motion
ho	String	Home
nl	String	Newline character if not ^j
up	String	Cursor up
<b>Screen Update</b>		
<i>Name</i>	<i>Data Type</i>	<i>Description</i>
al	String	Add blank line
ce	String	Clear to end of line
cd	String	Clear to end of display
cl	String	Clear screen
cs	String	Change scrolling region (VT100)
dc	String	Delete single character
dl	String	Delete line
dm	String	Enter delete mode
ed	String	End delete mode
ei	String	End insert mode
ic	String	Insert character
im	String	Insert mode
in	Boolean	Insert mode distinguishes nulls
ip	String	Insert pad after each character inserted
lk	String	Lock line
ua	String	Unlock all lines
uk	String	Unlock line
Bc	Numeric	Baud rate above which to ignore im, ic, and dc
Bl	Numeric	Baud rate above which to ignore dl and al
<b>Software Control of Terminal</b>		
<i>Name</i>	<i>Data Type</i>	<i>Description</i>
is	String	Initialization string
te	String	Ends programs for cursor motion
ti	String	Begins programs for cursor motion
ve	String	End open/visual mode
vs	String	Start open/visual mode
<b>Special Characteristics</b>		
<i>Name</i>	<i>Data Type</i>	<i>Description</i>
pc	String	Pad character if not NULL
tc	String	Entry for similar terminal
xc	Boolean	Cursor addressing relative to scroll region
xk	Boolean	Unsafe to move cursor with lines locked

## Basic Terminal Features

**co** indicates the number of columns on each line of the terminal screen. **li** gives the number of lines on the screen.

## Screen Movement Features

**bc** means that a character other than `Ctrl H` (the default) executes the backspace. Cursor motion, or **cm**, is discussed later in this section. **ho** takes a string data type that sends the cursor home, that is, to the upper left corner of the screen. **nl** indicates a newline character that is not `Ctrl J`. **up** tells how to move the cursor up a line in the same column on the screen.

**Cursor Motion:** Cursor motion enables you to move directly to any position on the screen. To initiate cursor motion, most terminals need an initial escape sequence, followed by the destination line, then the column. TERMCAP uses a variety of escape sequences, but line and column specifications are often the same for different terminals. On most terminals, lines and columns are not indicated by numbers, but by characters with ASCII values.

The following list shows the meanings of the `%` codings for cursor motion:

<i>Code</i>	<i>Description</i>
<code>%d</code>	Prints number as a decimal number.
<code>%2</code>	Prints number right justified as two digits with preceding 0 if needed.
<code>%3</code>	Prints number right justified as three digits with preceding 0's if needed.
<code>%. </code>	Outputs number as a character byte, coded in ASCII.
<code>%+x</code>	Adds the ASCII value of character <i>x</i> to value, then outputs the value as a character byte.
<code>%&gt;xy</code>	If value $> x$ , then value = value + <i>y</i> . Does not output anything.
<code>%r</code>	Reverses order of lines and columns so that column number is given before line number. No output.
<code>%i</code>	Increments line and column (for 1 origin). No output.
<code>%%</code>	Produces a single <code>%</code> .
<code>%n</code>	Exclusive-or row or column with 0140 (for DM2500). No output.
<code>%B</code>	BCD $(16*(x/10)) + (x\%10)$ . No output.
<code>%D</code>	Reverse coding $(x - 2*(x\%16))$ . No output (Delta Delta).
<code>%R</code>	<i>x</i> relative to <i>y</i> (i.e., $x = x - y + 1$ ). No output.

For example, the Ann Arbor Ambassador terminal uses a straightforward method of cursor addressing. To place the cursor on the 11th line, 11th column, the Ambassador requires the escape sequence `Esc [11;11H`. The cursor motion capability is as follows:

```
:cm=\E[%i%d;%dH:
```

The two `%d` strings are similar to the `printf()` statement in the C programming language, in that they indicate that the number is a digit. The `%i` means that line and column counts must be incremented. Ordinarily, they start with zero, but on the Ambassador, they begin with one.

The most common method of addressing screen positions is by adding the ASCII value of a space to the numbers of the line and column (starting at zero). In this case, the cursor motion string contains the code `%+space`. The ASCII value of space is 32 decimal. To move to line 11, column 11 (really line 10, column 10, because we started at zero), we add 10 to 32, producing 42, which is the decimal value of the ASCII character asterisk, `*`. For example, the Lear-Siegler adm3a terminal's cursor motion string is

```
:cm=\E=%+ %+ :
```

This string moves the cursor to line 10, column 10, with the escape sequence `Esc=**`.

The Digital Equipment VT52 and the Zenith h19 use a similar cursor motion string:

```
:cm=\EY%+ %+ :
```

On these terminals, the cursor moves to line 10, column 10, with `Esc Y**`. Most terminals use printing ASCII characters to indicate line and column location, but some use control characters. On these terminals, Control-@ would represent the first line or column. In this case, `%.` is used instead of `%+space`. The dot indicates that the given numeric value is used without adding the value of a space. When a terminal uses `%.`, it should be able to backspace the cursor and to move the cursor up one line (**up**).

Another example of cursor motion that shows padding is shown next:

```
:cm=6\E&a%r%2c%2Y:
```

To position to line 3, column 12, on the HP2645, the escape sequence is `Esc&a12c03Y`, padded for 6 milliseconds. The `%r` shows that the order of lines and columns is reversed. Note that the line and column numbers are printed as two digits.

## Screen Update Features

`al` indicates that if the cursor is at the first position on a line, the terminal can open a blank line before the current line. The cursor then appears on the new blank line. `ce` shows that the terminal can clear to the end of the line, leaving the cursor at its current position. `cd` means that the terminal can clear from the current position to the end of the display. `cl` means that the terminal can clear its screen.

To lock a portion of the screen, `cs` defines a scrolling region, or a subwindow of the screen that becomes active, thereby deactivating or effectively locking those lines outside of the scrolling region. The format of `cs` is the same as `cm`, except that the top line of the scrolling region is called `x`, and the bottom line is `y`. The definition for the VT100 is the following:

```
\E[%d;%dr
```

For **cs**, **%r** indicates that the number of lines is the second parameter for the scrolling region command, rather than the bottom line. **%r** must be the first encoding specified in **cs**.

**dc** deletes a single character while in delete mode. **dl** means that if the cursor is on the first position of a line, the terminal can delete the line. **dm** specifies delete mode, while **ed** indicates the exit delete mode.

**Insert mode:** Some terminals have an insert mode, while others send a sequence to open a blank position on the current line. **im** is the capability to get into insert mode (give **im** an empty value if your terminal inserts a blank position). **ei** is the capability to leave insert mode (give it an empty value also if you gave one to **im**). **ic** is any sequence to be sent just before inserting a single character. If your terminal has insert mode, you probably would not use **ic**, but you would use it for the terminal that opens a screen position. Use **ip** for post insert padding in milliseconds. **ip** may also include any other sequence that needs to be sent after inserting a single character.

Terminals handle insert/delete characters in two different ways. The most common method of inserting affects only characters on the current line and shifts characters off the edge of the screen at the end of the line. The other method, used by the Concept-100 and the Perkin-Elmer OWL, makes a distinction between typed and untyped blanks on the screen. When these terminals insert or delete, they shift only to an untyped blank on the screen. This blank is either eliminated or expanded to two untyped blanks.

To determine which type of terminal you have, clear your screen and type the following phrase, separated by cursor movements only:

```
asd  jkl
```

Do not type spaces between the groups of letters. Then, position the cursor before the **asd** and insert several characters. If the rest of the line shifts rigidly so that characters disappear off the end of the line, you have the first type of terminal, one that does not distinguish between blanks and untyped positions.

If the **asd** shifts over to the **jkl**, then both groups of letters move together, you have the second type of terminal. This terminal should have the capability, **in**, which means insert null. If your terminal does not fall into either of these two classes, you may have to change your **TERMCAP** entry.

The capabilities **lk** and **uk** lock and unlock specific lines. Both **lk** and **uk** are specified like **cm** where the affected line is the single parameter. It is assumed that locking or unlocking a line does not alter the current cursor position. For terminals that require the cursor to be positioned to the line to be locked or unlocked, the sequence must first save the current cursor position, then move it to the affected line, lock or unlock it, and finally restore the cursor position that was saved away. (If the terminal cannot save or restore cursor position, **lk** and **uk** should not be specified.) **ua** is the command to unlock all lines, which is preferable to unlocking all locked lines individually.

The numeric capabilities **Bc** and **Bl** specify the baud rate at or below which the terminal capabilities for insert/delete character and insert/delete line, respectively, are used. Baud rate is specified to EMACS via the **-SPEED** or **-BPS** option, and defaults to 9600.

## Software Control

`is` initializes the string for setting options. `ti` and `te` are used to enter and exit a special mode for cursor addressing, if your terminal requires it. Terminals that have more than one page of memory need this special mode. In order for cursor addressing to work properly on a terminal that has only memory relative cursor addressing (not screen relative cursor addressing), a screen-sized window must be fixed into the terminal. `vs` and `ve` are the start and end of sequences to change the cursor on the screen from an underline to a block and back. These sequences enhance cursor visibility while in EMACS.

## Special Characteristics

`pc` shows that the terminal requires a character other than an ASCII NUL as a timing pad.

`tc` indicates that two terminals are almost alike. It must be the last capability in the entry and is followed by the name of the similar terminal in the TERMCAP database. Because the TERMCAP entry is searched left to right, capabilities defined to the left of `tc` override the capability definitions for the similar terminal. You can also disable a capability by appending an `@` to its name in the definition.

For an example, refer to Figure 7-1, which is taken from the EMACS\*->TERM>TERMCAP file. Note that this example contains the capability `if`, which is not supported by Prime. (As noted earlier, other entries in the TERMCAP file may also contain capabilities not recognized by EMACS.)

Figure 7-1 illustrates several points:

- The capabilities to the left of `tc` override capabilities in the similar terminal (`vt100`).
- The `if` and `is` capabilities are turned off.
- `tc` is the last definition in the entry.

```
d0|vt100n|vt100 w/no init:is@:if@:tc=vt100:
```

**Figure 7-1**  
Using the `tc` Capability

`xc` indicates whether cursor positioning addresses are affected by the scrolling region. For example, if addressing is relative to the top line of the region, `xc` is specified. `xk` flags those terminals in which it is unsafe to perform cursor positioning while lines are locked on the screen, for example, the Prime PST 100.

## Control Characters, Delay Padding, and Placeholders

This section explains how to indicate control characters in capabilities that take a string data type, add padding characters, and use a placeholder to indicate the number of times that a command is to be repeated.

**Control Characters:** In capabilities that take a string data type, control characters are indicated either by an up-arrow (^) or by the string escape character (\). For example, ^c stands for `Ctrl C`, and \Ec maps to `Esc c`. The \ character may be produced by \\; the ^ character, by \^. To include a colon, put it in as an octal value by using the string escape character followed by the octal value (\272 or \072). A null character is \200.

The following list shows the escape sequences and their results:

<i>Escape Sequence</i>	<i>Result</i>
\E	Escape character
^	Control character
\nnn	Octal value of character
\n	Newline (^J)
\r	Return (^M)
\t	Tab (^I)
\b	Backspace (^H)
\f	Formfeed (^L)

**Delay Padding:** When the terminal is slow in performing a function, it needs padding characters for each affected line. The time delay in milliseconds is shown by an integer (for example, 10) or by an asterisk (\*) after the integer (10\*). The asterisk after the number indicates that 10 milliseconds of null padding will be repeated for each line affected by the operation. When the asterisk is used, it may be necessary to use the form "10.5" indicating a delay to tenths of milliseconds per unit. For example, the correct way to enter padding characters at the beginning of a string is `:d1=5*\ER:`. The asterisk after the numeral 5 shows that null padding is repeated for each deleted line.

**Placeholder Sequence:** Any string entry may contain a placeholder sequence indicating that the count for the number of times that command is to be executed may be included as part of the command in the indicated format. ANSI-compatible terminals support this notion, and the PST 100 entry (see Figure 7-3) shows many examples of its use. This sequence is supported in the following formats:

%d	Like "printf()" in C language.
%2	Prints number right justified in two spaces.
%3	Prints number right justified in three spaces.
%.	Outputs number as a character byte, coded in binary.
%+x	Adds the character x to value, then outputs the value as a character byte.
%%	Produces a single %.

This list is a subset of the formats supported for cursor addressing. As an example of how this information is used, to delete five lines on the CONCEPT-108, you would use the sequence `\E^B\E^B\E^B\E^B\E^B` or five `:d1=\E^B:` occurrences. To perform the same operation on the Prime PST 100, you would use `\E[5M`. This sequence is much more concise (and faster) than sending five `\E[1M` sequences. Thus, the PST 100 defines `:d1=\E[%dM:`.

## Adding an Entry to the TERMCAP Database

The Prime TERMCAP database resides in the file `EMACS*>TERM>TERMCAP`. This section tells you how to add your own entries to the TERMCAP file. It contains the following topics:

- A discussion of what you need to know before starting
- Two examples: a simple entry and a complex entry
- Testing a TERMCAP entry
- An alphabetical list of the TERMCAP capabilities (Table 7-2)

### Before Starting

Before adding entries to TERMCAP, you should have at hand the manual that describes your terminal's features. As you construct your entry, you may want to begin by using as a model an existing TERMCAP entry for a terminal that is similar to yours.

Put the entry in its own file so that you can test and, if necessary, revise it before you add it to the TERMCAP file. You can do this by using the `.TERMCAP$` global variable (discussed earlier) to specify the pathname for the file containing the entry you are working on.

#### Note

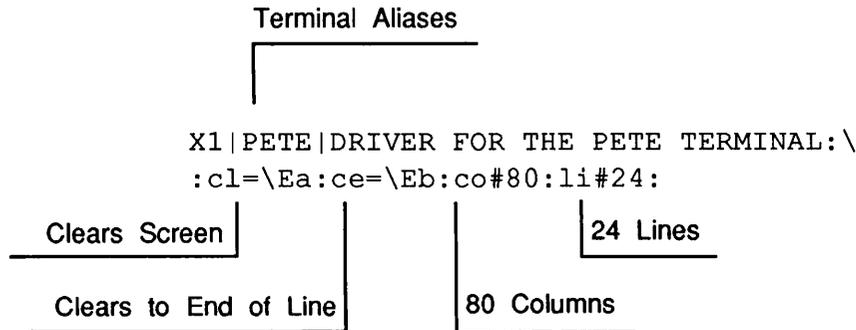
When preparing a new TERMCAP entry, if your terminal transmits `^J` for newline and `^H` for backspace, you do not need to define `nl` or `bc`, because your terminal uses the defaults for these capabilities.

### Building and Testing a TERMCAP Entry

You should observe the following guidelines as you begin building and testing a TERMCAP entry. Do not work directly in the TERMCAP file. Create a separate file to hold the working version of your entry. Then, when you are ready to begin testing, use the `.TERMINAL_TYPE$` and the `.TERMCAP$` global variables to specify the name of the terminal and location of entry to be tested. (See Global Variables above for a description of how to use these variables.) When you are satisfied that the entry is correct, have the System Administrator add it to the TERMCAP database. If this entry is to be used frequently, it should be located at the beginning of the file, because EMACS searches sequentially for an entry.

## Structure of Two Entries

An entry is readable as normal text, although it looks rather cryptic. Figure 7-2 shows the first two lines of an entry for a hypothetical terminal.



**Figure 7-2**  
**Two Lines of a Simple TERMCAPI Entry**

The first line is a list of aliases (names) for the terminal. The first two letters, X1, are a code for the terminal. They are followed by a common name for the terminal, PETE. A longer description of the terminal, DRIVER FOR THE PETE TERMINAL, completes the first line.

The first two names contain no blanks; however the description may contain blanks for readability. A vertical bar (|) separates terminal names, whereas a colon (:) ends the list of aliases.

If a TERMCAPI entry needs more than one line, the backslash character (\) at the end of the line shows that the entry continues to the next line. You should indent subsequent lines, for readability.

The second line is a list of two-letter capability names, followed by their arguments. Capabilities explain how the terminal accomplishes some well-defined task (like clearing the screen or moving the cursor). All capabilities are separated by colons. In this example, \E maps to an escape character.

The full entry for the Prime PST 100 terminal is shown in Figure 7-3. This particular entry may be outdated, and is used as an example only. Note that the PST 100 entry contains some TERMCAPI capabilities that are not supported by Prime and therefore are not documented in this chapter. Because EMACS ignores these capabilities, and recognizes only capabilities supported by Prime, including them in the entry causes no harm.



**Table 7-2**  
**Alphabetical List of Capabilities**

<i>Name</i>	<i>Type</i>	<i>Description</i>
<b>al</b>	String	Add blank line
<b>bc</b>	String	Backspace character if not ^H
<b>Bc</b>	Numeric	Baud rate above which to ignore im, ic, and dc
<b>Bl</b>	Numeric	Baud rate above which to ignore al, dl
<b>cd</b>	String	Clear to end of display
<b>ce</b>	String	Clear to end of line
<b>cl</b>	String	Clear screen
<b>cm</b>	String	Cursor motion
<b>co</b>	Numeric	Number of columns on screen
<b>cs</b>	String	Change scrolling region (VT100)
<b>dc</b>	String	Delete single character
<b>dl</b>	String	Delete line
<b>dm</b>	String	Enter delete mode
<b>ed</b>	String	End delete mode
<b>ei</b>	String	End insert mode
<b>ho</b>	String	Home
<b>ic</b>	String	Insert character
<b>im</b>	String	Insert mode
<b>in</b>	Boolean	Insert mode distinguishes nulls
<b>ip</b>	String	Insert pad after each character inserted
<b>is</b>	String	Initialization string
<b>li</b>	Numeric	Number of lines on screen
<b>lk</b>	String	Lock line
<b>nl</b>	String	Newline character if not ^J
<b>pc</b>	String	Pad character if not NULL
<b>tc</b>	String	Entry for similar terminal
<b>te</b>	String	Ends programs for cursor motion
<b>ti</b>	String	Begins programs for cursor motion
<b>ua</b>	String	Unlock all lines
<b>uk</b>	String	Unlock line
<b>up</b>	String	Cursor up
<b>ve</b>	String	End open/visual mode
<b>vs</b>	String	Start open/visual mode
<b>xc</b>	Boolean	Cursor addressing relative to scroll region
<b>xk</b>	Boolean	Unsafe to move cursor with lines locked

---

# 8

## Language Modes

### Introduction

Most programming languages have code formats that depend on indentation or specific column orientation. EMACS language modes make it easier for you to enter C, COBOL, FORTRAN, RPG, Common LISP, and PEEL programs in the correct format. While you are using any of these language modes, you can compile the code in your current buffer without leaving EMACS; if the program contains errors, EMACS displays diagnostic error messages and the corresponding faulty code.

For COBOL programs, the language mode changes some EMACS fundamental mode commands to simplify working with the column-oriented format. C mode changes some fundamental mode commands, while FORTRAN mode does not. VRPG mode, on the other hand, contains a few commands written specifically for use with VRPG programs. Common LISP mode contains all of EMACS fundamental mode functionality plus commands for Common LISP.

This chapter explains how to use the language modes. All commands and functions described in this chapter are found in the EMACS libraries.

#### Note

To turn on or off any of the language modes in this chapter, be sure to type `[Esc] [X]` first, then respond to the minibuffer prompt with the command name, for example, `cbl_on`.

### COBOL Mode

The COBOL language contains strict rules governing the formation of variables, statements, and paragraphs. COBOL also defines which columns must contain information. Consequently, COBOL mode changes the definition of certain fundamental commands so that they facilitate column-oriented entries.

#### Entering COBOL Mode

You must turn on COBOL mode using the `cbl_on` command. The `cbl_on` command specifies that the CBL compiler is invoked when you issue the compile command, as explained below. The CBL compiler is the only compiler available from COBOL mode.

**► cbl\_on**

This command turns on COBOL mode in the current buffer and sets the compile command to invoke the CBL compiler. After you type the `cbl_on` command, EMACS inserts six spaces at the beginning of each blank line in the file. It also resets the tabs to either the default COBOL tabs or to tabs that you have predefined in a library file.

## Entering Text in COBOL Mode

**Tabs:** The default tabs for COBOL mode are set at the following columns:

```
8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
```

If you want different tab stops, you can initialize the `my_cobol_tabs$` variable by adding a statement such as the following to a library file:

```
(setq my_cobol_tabs$ "8 12 20 28 36 72")
```

Unlike the other variables you can set, these tab stops will not become active until you enter COBOL mode. If you want them immediately, you can use one of the normal fundamental mode tab commands.

## Compiling COBOL Programs

You can compile a COBOL program from EMACS using the compile command. Refer to the Additional Information About Compiling and Debugging Programs section later in this chapter for instructions about using this command.

## Exiting from COBOL Mode

Use the following command to turn off COBOL mode in the current buffer.

**► cbl\_off**

## Additional COBOL Mode Information

When you invoke COBOL mode, EMACS modifies its definition of certain commands. For example, in fundamental mode, `Ctrl B` moves point backward one character. In COBOL mode, `Ctrl B` still moves point backward one character, but if point is at column 7, it is moved to the last character on the previous line. That is, in COBOL mode, column 7 is considered to be the beginning of the line, not column 1.

The following keybindings and EMACS extended commands have modified definitions in COBOL mode.

▶ **Ctrl A** **cobol\_begin\_line**

This command moves point to column 7 instead of column 1.

▶ **Ctrl B** **cobol\_back\_char**

This command moves point back one character.

▶ **Ctrl D** **cobol\_delete\_char**

Deletes the character on which the cursor is positioned. If the cursor is at the end of a line, it deletes the carriage return and *all* leading spaces on the next line. If this newly created line is longer than 72 characters, it is reformatted automatically so that no characters appear after column 72.

▶ **Ctrl E** **cobol\_end\_line**

This command moves to the end of the line. If, for some reason, the end of the line is after column 72, the line is reformatted so that no characters appear after column 72.

▶ **Ctrl F** **cobol\_forward\_char**

This command moves point forward one character. If point is at the end of the line, point moves to column 7 of the next line.

▶ **Ctrl H** **cobol\_rubout\_char**

This command deletes the character to the left of point. If point is in column 7, the initial blanks at the beginning of the line are removed as well as the carriage return separating the current line and the previous line. If the resulting line is longer than 72 characters, it is automatically reformatted.

▶ **Backspace** **cobol\_rubout\_char**

This command is identical to **Ctrl H** described above.

▶ **Return** **cobol\_wrap**

In COBOL mode, if you type **Return** at the end of a line longer than 72 characters, EMACS inserts a carriage return. EMACS tries to produce appropriate indentation on the second line. In most cases, EMACS indents to the same position as the previous line. However, if the previous line began in column 8, the second line indents to column 12. If the previous line is blank, the second line indents to column 8.

▶ **Ctrl K** **cobol\_kill\_line**

This command kills the current line. If point is at the end of the line, it kills the carriage return separating the two lines. If the resulting line is more than 72 characters long, the line is reformatted. Note that if point is around column 72, this command does not allow the kill to occur. You have to go to the next line and kill it.

▶ **Ctrl O** **cobol\_open\_line**

This command performs the same action as it does in fundamental mode except that all lines that are created are padded with spaces to column 7.

▶ **Ctrl W** **cobol\_kill\_region**

This command performs the same action as it does in fundamental mode except that it ensures that the current line is no longer than 72 columns after the region has been killed.

▶ **Ctrl Y** **cobol\_yank\_region**

This command performs the same action as the fundamental mode command except that it ensures that text is inserted between columns 7 and 72.

▶ **Ctrl X Ctrl H** **cobol\_backward\_kill\_sentence**

This command kills all text from point back to the end of the previous sentence. Unlike the fundamental mode command, a period is the only recognized terminator.

▶ **Ctrl X Ctrl K** **cobol\_backward\_kill\_line**

This command kills all text from point back to the beginning of the current line. It then inserts six spaces so that point is in column 7.

▶ **Ctrl X Ctrl R** **cobol\_read\_file**

This command performs the same action as in fundamental mode except that it inserts six spaces at the beginning of all blank lines.

▶ **Ctrl X Ctrl Z <** **cobol\_mark\_top**

This command performs the same action as it does in fundamental mode except that a mark is put in column 7 rather than column 1.

▶ **Ctrl X** **Ctrl Z** **>**

**cobol\_mark\_bottom**

This command performs the same action as it does in fundamental mode except that it ensures that a mark is put in column 7 if the final character in the region is a carriage return. If this last character is a carriage return, it pads the line with six spaces.

▶ **Ctrl X** **R**

**cobol\_repaint**

This command performs the same function as it does in fundamental mode except that when point is moved to a line, it is placed in column 7 rather than column 1.

▶ **Ctrl X** **[**

**cobol\_back\_para**

This command moves point backward one paragraph. In COBOL mode, a paragraph is any line that has text in columns 8 through 11. If the text in these columns is a comment, the line is skipped.

▶ **Ctrl X** **]**

**cobol\_forward\_para**

This command moves point forward one paragraph, that is, any line that has text in columns 8 through 11. If the text in these columns is a comment, the line is skipped.

▶ **Esc** **Ctrl H**

**cobol\_rubout\_word**

This command deletes the word preceding point. In COBOL mode, a hyphen is considered part of a word. (In fundamental mode, a hyphen is a word separator.) **Ctrl H** is usually the Backspace key.

▶ **Esc** **<**

**cobol\_move\_top**

Moves point to column 7 of the first line in the buffer.

▶ **Esc** **>**

**cobol\_move\_bottom**

This command moves point to the end of the last line in the buffer. If the last line is a blank line, point is put in column 7.

▶ **Esc** **A**

**cobol\_backward\_sentence**

This command moves point backward to the end of the previous COBOL statement. The end of the statement is defined by a period.

▶ **[Esc] [B]** **cobol\_back\_word**

This command moves point back one word. Unlike fundamental mode, a hyphen is considered part of a word in COBOL mode.

▶ **[Esc] [D]** **cobol\_delete\_word**

This command deletes the word following point. If point is at the end of the line, COBOL mode checks to see if the new line is longer than 72 characters. If it is, it reformats the line.

▶ **[Esc] [E]** **cobol\_forward\_sentence**

This command moves point forward to the beginning of the next COBOL statement, placing the cursor on the character following the first period (.) it encounters.

▶ **[Esc] [F]** **cobol\_forward\_word**

This command moves the cursor forward one word. It places the cursor on the first word separator (space or punctuation mark) following the end of the word. Hyphens are not word separators in COBOL mode.

▶ **[Esc] *digit* [Esc] [G]** **cobol\_goto\_line**

This command moves point to the line number specified by "digit". If you do not specify a number, the command moves point to the first line in the buffer. It differs from fundamental mode in that it ensures that point is not before column 7.

▶ **[Esc] [V]** **cobol\_white\_delete**

This command deletes the space characters that surround point. It ensures that space characters remain in columns 1 through 6.

▶ **[SPACE]** **cobol\_wrap**

This command works in the same manner as inserting a space in fill mode, but it does not affect text in columns 1 through 6.

The following commands are disabled in COBOL mode:

▶ **[Ctrl X] [Ctrl L]** **lowercase\_region**

▶ **[Ctrl X] [Ctrl Z] [Ctrl A]** **backward\_clause**

▶	<code>Ctrl X</code> <code>Ctrl Z</code> <code>Ctrl E</code>	<code>forward_clause</code>
▶	<code>Ctrl X</code> <code>Ctrl Z</code> <code>Ctrl H</code>	<code>backward_kill_clause</code>
▶	<code>Ctrl X</code> <code>Ctrl Z</code> <code>Ctrl K</code>	<code>forward_kill_clause</code>
▶	<code>Ctrl X</code> <code>Ctrl Z</code> <code>S</code>	<code>center_line</code>
▶	<code>Esc</code> <code>L</code>	<code>lowercase_word</code>
▶	<code>Esc</code> <code>Q</code>	<code>fill_para</code>

If a command does not appear in either of the preceding lists, its definition is the same in COBOL mode and fundamental mode.

## C Mode

C mode provides an environment that helps the user enter and edit C programs by offering features that improve the syntax and structure of programs. The syntax aids consist of templates for a number of standard language structures. The pretty printer checks the program syntax and produces error messages for incorrect syntax, in addition to formatting the program text to reveal its structure. You can pretty print all or part of a program. Pretty printing is done automatically with the templates for some program constructs and by user request for the rest of the program.

### Entering C Mode

Use the following command to turn on C mode in the current buffer.

#### ▶ `cc_on`

The buffer must contain a C source program when you issue this command, or it must be an empty buffer that you will use to enter new C source code. The accepted suffix for a file or buffer containing a C program is `.C` or `.CC`. You can move freely between buffers where C mode is turned on or off.

Another way to invoke C mode is to use the file hooks feature that is described in Chapter 6 of this book. Your `user_type$` variable must be set to "programmer\$", so that the local file hook automatically activates C language mode when you select a file or buffer with a `.C` or `.CC` suffix. You may set up user-defined keybindings to activate C mode.

When you invoke C mode, it reads in an EMACS speed-type file that contains templates for the C language forms. You may see a list of the speed-type abbreviations for the C templates, as well as other speed-type definitions that you may have created, by typing the `spd_list_all` command. See Chapter 5 in this book for more information about speed-type commands.

If you are entering a new program into an empty buffer, use the abbreviation `/prog` to produce a template that contains global and local declarations. You will notice two comments near the top of the skeleton C program:

```
/* GLOBAL DECLARATIONS */
```

and

```
/* LOCAL DECLARATIONS */
```

Do not delete or alter these comments in any way if you intend to use C mode's declaration/definition management commands. The comments are "landmarks" for C mode to mark locations in your source code. If your C source code does not contain these comments, it is not possible for you to use C mode's declaration/definition commands.

You can set the number of spaces for indenting by using the `/for` command, which indents the text two spaces.

You can ask for the whole program or any part of the program text that contains a complete function to be pretty printed and checked for syntax by marking the appropriate region and using the `pp_region` command that invokes the pretty printer. Note that C language mode's pretty printer operates only within a defined region. If the region does not contain a syntactically self-contained object, the pretty printer reports a syntax error, because it does not see any portion of the buffer that lies outside of the region. The `pp_region` command is bound to `Ctrl X \`. When you give this command, the pretty printer invokes the parser and gets back a list of pretty printer actions sorted by line. It uses this action list to pretty print the text.

For other pretty printing, for example, a single if-else statement, use the `Ctrl X L 1` keystroke.

## Exiting from C Mode

Use the following command to turn off C mode in the current buffer: It is not necessary to turn off C mode before exiting from EMACS.

► `cc_off`

## Commands

There are two general classes of C mode commands:

- Declaration/definition management
- Source/appearance management (pretty printing)

C mode makes it possible for you to move around in the EMACS buffer to add declarations and definitions while you are writing the program. The pretty printing feature formats all or part of the program text to reveal its structure.

The C mode changes certain fundamental mode keybindings, as shown in the following list of keybindings and commands for C mode. The list includes extended commands that are not bound to keypaths.

► **cc\_on**

This command turns on the C language mode in the current buffer and sets the compile command to invoke the C compiler. It checks to see whether the C mode speed-type definitions have been loaded; if not, an error message appears in the minibuffer. It tells the user to type `Ctrl X ?` for help.

► **cc\_off**

This command turns off the C language mode in the current buffer.

► `Ctrl X \` **pp\_region**

This command does syntax checking and pretty prints an EMACS region if the region contains a complete program or the complete body of a C function. If a syntax error is found, EMACS provides the following error message and leaves the cursor on the line where the syntax error appears.

```
Syntax error at current cursor
```

The initial left margin is set at the first non-whitespace character to the right of the cursor.

► `Ctrl X L 1` **cc\_pp\_function\$**

This command pretty prints a function or definition that is not part of a function body, for example, a single if-else statement.

► `Ctrl X G` **cc\_add\_global\_def\$**

This command prompts for a global declaration by searching back to the string `/* GLOBAL DECLARATIONS */` and displaying the following placeholders.

```
<.type.> <.identifier.> ;
```

► `Ctrl X C` **cc\_add\_local\_def\$**

This command prompts for a local declaration in the current function. It searches back to the nearest `/* LOCAL DECLARATIONS */` header and displays the following placeholders.

```
<.type.> <.identifier.> ;
```

▶ `Ctrl X /` `cc_ret_from_def$`

This command returns the cursor to the place in the text where either the `cc_add_local_def$` or the `cc_add_global_def$` command was invoked.

▶ `Ctrl X ?` `cc_help$`

This command displays the C help commands and abbreviations on your screen.

▶ `Return` `cpret_indent_relative`

This command inserts a carriage return into the text and moves the cursor to the beginning of a new line. The new line is indented to the first non-whitespace character of the previous line.

## Abbreviations and Templates

One of the most useful features of the C language mode is its ability to create accurate speed-type templates for the major language forms in C. When you cause a template abbreviation to expand by typing the abbreviation, followed by a valid separator character (a space or punctuation mark), the cursor rests on the `<` symbol at the beginning of the first placeholder in the template. The next character that you type erases the placeholder. Use the standard EMACS commands, `Ctrl X >` (`forward_place_holder`) and `Ctrl X <` (`back_place_holder`), to move the cursor to the next or previous placeholders. If there is no placeholder in the template, the cursor is placed immediately following the template. If you need help using speed-type templates, see the information about speed-type templates and placeholders in Chapter 5.

The following list shows the speed-type abbreviations and their templates that are available for EMACS C mode.

▶ `/prog`

```
/* GLOBAL DECLARATIONS */
<.type.> <.identifier.>;

<.type.> <.name.> (<.arg_identifier.>)
<.type.> <.arg_identifier.>;
{
    /* LOCAL DECLARATIONS */
    <.type.> <.identifier.>
    <.statement.>
}
```

**▶ /func**

```
<.function_name.> (<.arguments.>)  
<.type.> <.argument.>  
{  
    /* LOCAL DECLARATIONS */  
    <.type.> <.identifier.>  
    <.statement.>  
}
```

**▶ /if**

```
if (<.expression.>)  
    <.statement.>
```

**▶ /ifelse**

```
if (<.expression.>)  
    <.statement.>  
else  
    <.statement.>
```

**▶ /while**

```
while ( <.expression.> )  
    <.statement.>
```

**▶ /do**

```
do  
    <.statement.>
```

**▶ /for**

```
for ( <.init_expr.> ; <.condition.> ; <.step_expr.> )  
    <.statement.>
```

**► /switch**

```
switch ( <.expression.> )
{
  case <.constant_expr.> :
    <.statement.>
  default :
    <.statement.>
}
```

**► /case**

```
case <.const_expression.>
  <.statement.>
```

**► /default**

```
default <.const_expression.>
  <.statement.>
```

**► /struct**

```
struct <.identifier.>
{
  <.type.> <.identifier.>
}
```

**► /union**

```
union <.identifier.>
{
  <.type.> <.identifier.>
}
```

**► /dcl**

```
<.type.> <.identifier.>;
```

**▶ /block**

```
{  
  <.statement.>  
}
```

**▶ /body**

```
{  
  <.statement.>  
}
```

**▶ /{**

```
{  
  <.statement.>  
}
```

**▶ //**

```
/* <.comment.> */
```

Comments are treated as whitespace and ignored by the pretty printer.

## Additional C Mode Information

**Preprocessor Directives:** The C language contains a class of constructs that are directives to the C language preprocessor. These constructs begin with the # symbol. The C language mode treats the entire class of preprocessor directives as whitespace and ignores them when the pretty printer is called.

**Configuration:** PEEL code for the C language mode is in EMACS\*>EXTENSIONS>CC.EFASL and EMACS\*>EXTENSIONS>SOURCES>CC.EM. Speed-type abbreviations for C mode are in the file EMACS\*>EXTENSIONS>SPD>CC.ESPD and EMACS\*>EXTENSIONS>SPD>CC.SPDT. The pathname of the C mode parser that is used to output pretty print commands is EMACS>LIBRARIES\*>CC\_LM.RUN.

## FORTRAN Mode

The major function of FORTRAN mode is to automatically indent the text that you type, depending on the character you enter in column 1 of a new line.

FORTRAN mode does not redefine any of the fundamental commands. All fundamental commands exist in FORTRAN mode and work in the normal way.

## Entering FORTRAN Mode

The commands for entering FORTRAN mode are discussed below.

### ► `fortran_on`

This command turns on FORTRAN mode in the current buffer. If your current file has a `.F77` suffix, the following message appears:

```
FORTRAN mode now on, language F77
```

If your current file has a `.FTN` suffix, the message is

```
FORTRAN mode now on, language FTN
```

## Entering Text in FORTRAN Mode

**Tabs:** The default tab stops in FORTRAN mode are:

```
7 13 19 25 31 37 43 49 55 61 67 72
```

If you want different tab stops, you can initialize the `my_fortran_tabs$` variable by adding a statement such as the following to a library file:

```
(setq my_fortran_tabs$ "7 17 27 37 47 57 67 72")
```

Unlike the other variables you can set, these tab stops will not become active until you reenter FORTRAN mode. If you want them immediately, you can use one of the normal fundamental mode tab commands.

**Column 1 Commands:** The actions EMACS takes in FORTRAN mode are controlled by special characters that you type in column 1 of an empty line. If you type any of these characters in other columns, they are inserted normally. These characters are described below.

- If you type a slash (/) or an asterisk (\*) in column 1 on an empty line, EMACS assumes that you are inserting a comment line. It puts a C in column 1 and indents to column 3. (This column can be changed. See below.)
- If you type an ampersand (&) in column 1, EMACS inserts a C in column 6 and then enters spaces so that the current line is indented the same as the previous line. Labels and intervening comments are ignored. (The character typed in column 6 can be changed. See below.)

- If you type a number in column 1, EMACS assumes that you are typing a label and just inserts the number. When you are done typing the label, use the tab or space keys to move to where you want the text to be entered.
- Any other typed character is automatically indented the same as the previous line. For example, if you type the letter *B* in column 1 and the previous line had a statement that began in column 18, EMACS inserts the *B* in column 18 of the current line. EMACS ignores intervening comments.

## Compiling FORTRAN Programs

You can compile a FORTRAN program from EMACS using the compile command. Either the F77 compiler or the FTN compiler is invoked, depending on the file suffix. You must use one of these two suffixes and you must have the correct suffix for the language you are using. If your program has no suffix, or an incorrect suffix, the compiler is not invoked. Refer to the Additional Information About Compiling and Debugging Programs section later in this chapter for detailed instructions about using this compile command.

## Exiting from FORTRAN Mode

Use the following command to turn off FORTRAN mode in the current buffer:

► `fortran_off`

## Additional FORTRAN Mode Information

FORTRAN mode has some other useful commands. These are discussed below.

► `fortran_do_toggle`

Some users like to add a level of indentation after they type a DO statement. This command controls that indentation. The first time you type this command, it tells EMACS to indent the text you type after a DO statement. The second time you type it, it tells EMACS that you no longer want indentation to occur.

You can also add this statement to a library file by setting the `fortran_do$` variable to true.

► `set_fortran_comment`

Normally in FORTRAN mode, EMACS puts a C in column 1 and indents to column 3 after you type a / or \* in column 1.

This command changes the comment indentation to the value you specify. When you type it, EMACS prompts you for a value.

You can also control comment indentation by setting the `fortran_comment_indent$` variable in a library file.

**► set\_fortran\_cont**

Normally, EMACS uses a C as the continuation character. If you wish to use a different character, you can type this command and EMACS will prompt you for the character you want to use.

You can also specify this character by setting the `fortran_continue_char$` variable in a library file. Note that a source code line ending in column 72 does not generate a continuation line.

## RPG Mode

RPG is a structured language that requires adherence to a rigid format for entering source code. Traditionally, coding a program in VRPG involves using a variety of specification sheets and placing fields carefully in specific columns, following the formats of the specification sheets. The lines of code are then keyed into the computer. EMACS RPG mode makes the entry of source code easier by providing specification sheet templates online.

### Entering RPG Mode

Use the following command to enter RPG mode:

**► vrpg\_on**

This command turns on RPG mode in the current buffer. It tells EMACS to use the VRPG compiler whenever you invoke the compile command. As soon as RPG mode is invoked, the screen is cleared and one of the specification sheet templates is displayed on the first four lines at the top of the screen. The type of template that is displayed depends on the entry in column 6: if there is no entry in column 6, EMACS displays the header template with an H in column 6. The header template is the default template.

**Specification Sheet Templates:** Figure 8-1 shows the seven RPG-mode specification sheet templates as they appear on your screen. Each template shows the layout of the current specification and indicates the names of the fields. The seven templates are as follows:

- Header template displays an H in column 6
- File template displays a F in column 6
- Extension template displays an E in column 6
- Line counter template displays an L in column 6
- Input template displays an I in column 6
- Calculation template displays a C in column 6
- Output template displays an O in column 6



**Modifying Template Appearance:** Instead of using the fully formatted template, many experienced programmers may prefer a simpler template that labels only columns. To select this version, use the `rpg_config` command, which is explained in a later section. The abbreviated template (ruler template) looks like this:

```

          1         2         3         4         5         6         7         8
1234567890123456789012345678901234567890123456789012345678901234567890

```

### Ruler Template

## Entering Text in RPG Mode

In RPG mode, the screen displays a *shadow cursor*, indicated by a caret (^), in addition to the text cursor. The shadow cursor is located on the line immediately under the template. It moves as the cursor moves, allowing you to determine at a glance in which column you are making an entry.

Most of the fundamental mode EMACS commands, such as `Ctrl F`, are not changed in RPG mode. However, three keypaths — `Ctrl B`, `Esc B`, and `Backspace` — move the cursor back only as far as the first column in the present line. Use any logical sequence of EMACS commands (such as `Ctrl Z` followed by `Ctrl E`) to move to column 80 of the previous line.

RPG mode provides two special keybindings that move the cursor from field to field. They are the `Tab` key and the `Return` key.

▶ `Tab`

`rpg_tab$`

Pressing the `Tab` key moves the text cursor to the next field of the current specification sheet. For example, if you are at column 15 of an input (I) specification, pressing `Tab` puts the text cursor at column 19. Column 19 is the first column in the next field, which is for record indicators.

The `Tab` key moves forward only to fields on the current line; to get to the next line, you must use `Ctrl N` or the `Return` key.

▶ `Return`

`rpg_cr$`

Pressing `Return` inserts a blank line between the current line and the next line and puts the text cursor in column 6. This feature is convenient both for writing new programs and for editing old ones. However, you must be careful if you are editing a program. If you press `Return` while the cursor is over text, two things happen:

- The rest of the current line is placed on a new line, starting at column 1.
- The cursor moves to column 6.

## Compiling VRPG Programs

You can compile a VRPG program from EMACS, using the `compile` command. All of the standard command line options for compiling a VRPG program are supported, but they must be entered in uppercase. If there are errors in the program, EMACS provides a message at the bottom of the screen, indicating the number of errors, the command to view the next error, and the command to view the previous error. For information about compiling, see the Additional Information About Compiling and Debugging Programs section at the end of this chapter.

## Exiting from RPG Mode

Use the `vrpg_off` command to turn off RPG mode in the current buffer.

## Additional RPG Mode Information

RPG mode provides its own set of EMACS commands and configuration variables. As with other EMACS commands, you can execute these commands in any of the following ways:

- Enter the keybinding, if the command is bound to a keypath.
- Enter the extended command, after keying `[Esc] [X]`. Respond to the displayed query **Command:** with the name of the function, such as `rpg_template$`.
- Use a Prime EMACS Extension Language (PEEL) macro. Using PEEL statements will probably be most useful to you as a way to put commands and variables into your user library.

This section lists RPG mode commands, grouped according to the following functional categories:

- Configuring RPG mode
- Getting help
- Modifying the template
- RPG library files

For each command, the keybindings are listed first and the extended commands are listed second. The word *none* indicates that there are no set keybindings.

**Configuring RPG Mode:** The `rpg_config` command provides a series of options to configure RPG mode.

### ► `rpg_config`

The options provided by this command select the terminal screen appearance and the compiler. Some of these options can be overridden later by the other RPG mode commands.

Invoking this command provides a series of questions in the minibuffer. The first display looks like this:

```
Enable automatic display of template (current value -YES):
CR - no change; (Y YES T TRUE) - set true; (N NO F FALSE) - set false;
```

The display for each question shows the current value of the variable and the choices of responses. You can either retain the current value or change it. For example, if you enter **Y** or press `[Return]` in response to the query above, the appropriate template is automatically displayed.

Figure 8-2 diagrams the `rpg_config` facility selections, with the default values highlighted. You can override some of the individual configuration conditions by using other RPG mode commands. For example, if you have shut off the automatic display of templates, use `[Esc] [X]` `[Ctrl C]` to view the current template. Then, to delete the template, invoke `[Esc] [Ctrl K]`.

**Getting Help:** The top line of the minibuffer contains `rpg` and the word **overlay**, indicating overlay mode. The minibuffer can also display help messages. If you want to see them, you must select this option when you configure RPG mode, as explained in the next section. The help messages are either in a full or in an abbreviated form. The full form shows:

- The position of the cursor
- Descriptive information for that column or field

For example, if the cursor is in column 12 of the calculation specification, the full form help message looks like this:

```
12 col 12 Indicator Specifier-N for not Indicator
```

The abbreviated (partial) form of the same help message displays the current column number at the leftmost side of the minibuffer:

```
12
```

There are two commands that override configuration directives and give you access to the `rpg_config` facility help messages. The choice depends upon how you configured RPG mode.

► `[Esc] [H]` `rpg_full_help$`

If you configured the partial help message, this command provides a full help message that is based on the current specification sheet and column number. This command does not override the `rpg_config` setting for displaying a help message.

► `rpg_help$`

If you did not configure a help display, this command gives a partial help message in the minibuffer.

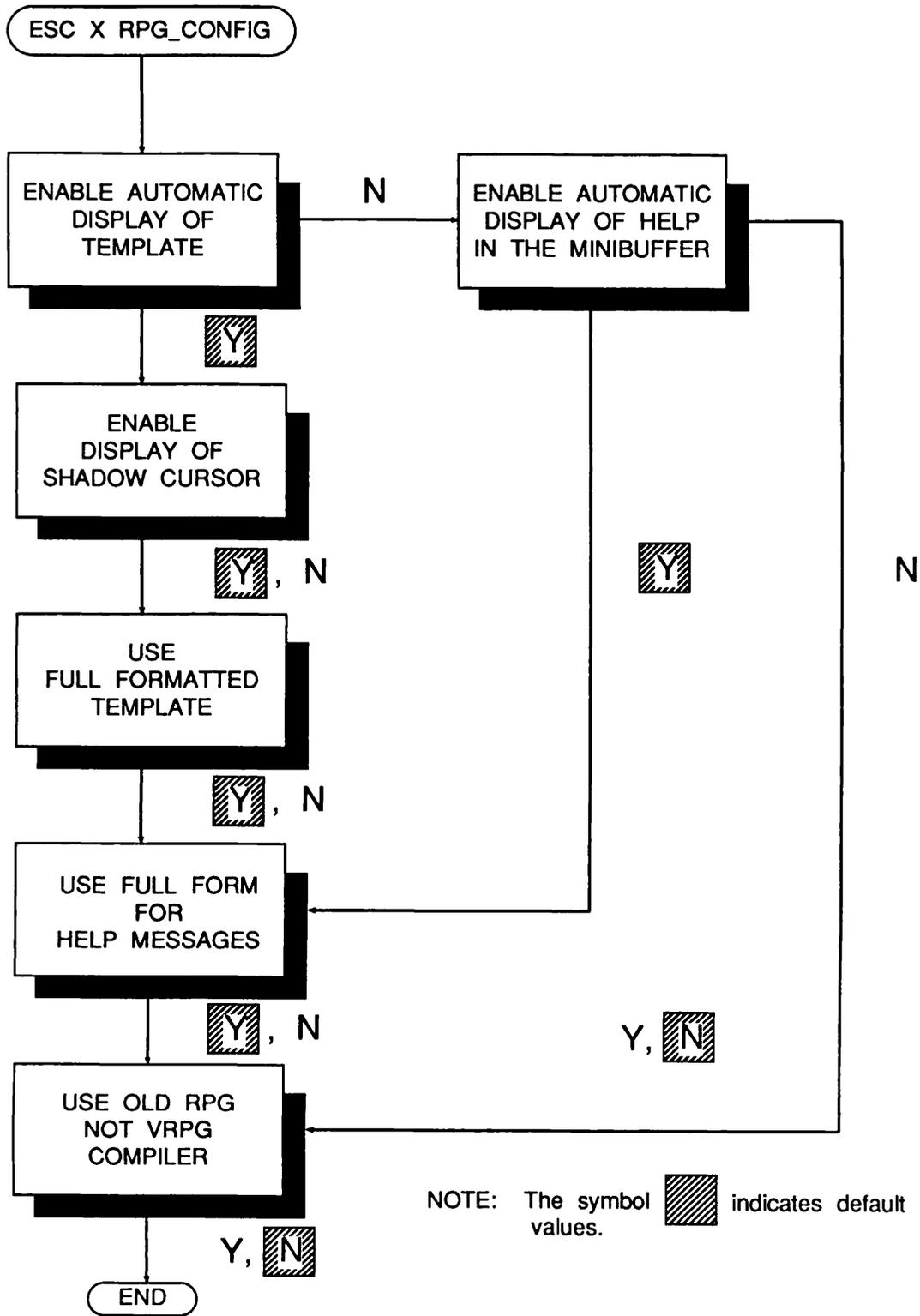


Figure 8-2  
The RPG Config Selections

**Modifying the Template:** Two more commands override settings made through the `rpg_config` facility. The first makes an RPG template appear on the screen; the second removes the template from the screen.

► `[Esc] [H]` `rpg_full_help$`

► `[Esc] [Ctrl C]` `rpg_template$`

This command displays an RPG template based on the specification form specifier in the current line. If there is no card specifier in the current line, `[Esc] [Ctrl C]` first prompts you to enter one, and then displays the template.

This command is useful if you have shut off the automatic display of templates with the `rpg_config` facility but still want to see them occasionally.

► `[Esc] [Ctrl K]`

This command removes the RPG templates from your screen. It overrides the `rpg_config` setting for displaying the templates. After you enter this command, the minibuffer displays the following message:

```
Use rpg_template$ to restore the template window.
```

**RPG Library Files:** EMACS RPG mode lets you set values in a library file to establish keybindings and performance. You may find it convenient to use the statement that enables the file hooks mechanism:

```
setq user_type$ 'programmer$
```

This command should be the last command in your library. For more information about setting keybindings, library files, and file hooks, see Chapter 6.

You can store two types of variables in your user library:

- Keybinding variables
- `rpg_config` variables

Each type is discussed in this section.

The first type of RPG variable for your user library is the keybinding variable. Normally, RPG mode uses the following four bindings:

<i>Keypath</i>	<i>Extended Command</i>	<i>Variable</i>
<code>[Ctrl I]</code> or <code>[Tab]</code>	<code>rpg_tab\$</code>	<code>rpg_tab_key\$</code>
<code>[Esc] [H]</code>	<code>rpg_full_help\$</code>	<code>rpg_help_key\$</code>
<code>[Esc] [Ctrl C]</code>	<code>rpg_template\$</code>	<code>rpg_plate_key\$</code>
<code>[Esc] [Ctrl K]</code>	<code>rpg_kill_template\$</code>	<code>rpg_unplate_key\$</code>

If you are not using the Standard User Interface (SUI), you may bind these settings to function keys. In SUI, these commands are already automatically bound to the terminal's function keys.

To rebind a keypath, put the `setq` statement for the new keypath in your library. The following is an example of binding the `rpg_template$` command to a function key whose keypath is `Esc [ 4 z`.

```
(setq rpg_template$ "^[[4z")
```

This statement can go anywhere in a library file.

The second type of RPG variable for your user library is the `rpg_config` variable. When you use the `rpg_config` facility, EMACS sets a variable to either true or false, depending on your answer. If you always want RPG mode to perform in a way different from the default, you can place these same variables in a library startup file.

If you configure a default, you can still change the way in which RPG mode performs in the current session by using `[Esc] [X] rpg_config`. However, any changes made will only be used in the current editing session. At your next editing session, your settings will again be used. You can place these statements anywhere in a library file.

Table 8-1 shows the RPG mode variables, their default values, and a description of the action controlled by the variable.

**Table 8-1**  
**RPG Mode Variables**

<i>Variable</i>	<i>Default</i>	<i>Meaning</i>
<code>rpg_split_window\$</code>	true	This variable indicates whether the mode will display information in a window at the top of the screen. The kind of information is described by the <code>rpg_full_template\$</code> variable. If this variable is false, EMACS disregards the <code>rpg_full_template\$</code> variable.
<code>rpg_full_template\$</code>	true	This variable indicates the kind of information that will appear in the top window. If this variable is set to true, EMACS places a template in the top window. This template changes according to the specification form. If it is set to false, only a ruler will appear in the window.
<code>rpg_show_shadow\$</code>	true	This variable controls the use of the shadow cursor, which shows the current column position. If this variable is set to false, EMACS does not use the shadow cursor.
<code>rpg_help_sw\$</code>	true	This variable indicates whether EMACS will show help information in the minibuffer area. The kind of information is defined by the <code>rpg_full_help_sw\$</code> variable. If this variable is set to false, EMACS ignores the <code>rpg_full_help_sw\$</code> variable.
<code>rpg_full_help_sw\$</code>	true	This variable tells EMACS what kind of help information to print. Help information is only printed when <code>rpg_help_sw\$</code> is set to true. If this variable is set to true, EMACS displays the current column, the limits of the current field, and a brief description of the information you can type in the field. This information depends on the specification form. If this variable is set to false, EMACS prints only the current column number.

## LISP Mode

LISP mode is one of the three LISP-related language modes provided by EMACS. The other two are Common LISP mode and Listener mode. LISP mode is designed for use with PEEL, Prime's EMACS Extension Language, with which you can write new EMACS commands, called *extended commands*. See *The EMACS Extension Writing Guide*. Although PEEL resembles LISP in syntax and contains many of LISP's basic list-processing functions, it is not equivalent to PRIME Common LISP, and LISP mode is not equivalent to Common LISP mode. Common LISP mode contains all of LISP mode's functionality.

### Entering LISP Mode

The following command turns on LISP mode in the current buffer:

► `lisp_on`

Once LISP mode is invoked, `lisp` appears on the EMACS status line.

### Exiting from LISP Mode

The following command turns off LISP mode in the current buffer:

► `lisp_off`

### Additional LISP Mode Information

The following keybindings and EMACS extended commands are effective only in LISP and Common LISP modes:

► `[Esc] [Ctrl B]` `balbak`

This command moves the cursor, or point, backward to the opening parenthesis of the current level of nesting. If point is at a closing parenthesis, EMACS moves point to the corresponding opening parenthesis. If point is between forms, EMACS moves point back to the last closing parenthesis and then back again to the corresponding opening parenthesis.

► `[Esc] [Ctrl F]` `balfor`

This command moves point forward to the closing parenthesis of the current level of nesting. If point is at an opening parenthesis, EMACS moves point to the corresponding closing parenthesis. If point is between forms, EMACS moves point forward to the next opening parenthesis and then forward again to the corresponding closing parenthesis.

▶ `)` `close_paren`

This command moves point briefly to the opening parenthesis that corresponds to the closing parenthesis just typed. After a pause, EMACS returns point to the position immediately following the closing parenthesis just typed. The pause at the opening parenthesis usually lasts 750 milliseconds. If you want to change the default duration of the pause, set the EMACS variable `lisp.paren_time` to the new value. A value of 0 turns off `close_paren`.

▶ `Return` `lisp_cr`

This command pretty prints your code as you type it. Pretty-printing arranges the code so that indentation is consistent and the structure of expressions is clear.

▶ `Esc Q` `lisp_pp_command`

This command pretty prints the current form, starting at point.

▶ `Esc ;` `lisp_comment`

This command moves point to the LISP comment column, usually column 40, and places a semicolon (;) there. If you want to change the default LISP comment column, set the EMACS variable `lisp_comment_column` to the new value.

▶ `Esc I` `lisp_indent`

This command indents point to the proper position for the current LISP form.

▶ `Esc Ctrl A` `begin_defun`

This command moves point backward to the last opening parenthesis against the left margin.

▶ `Esc Ctrl E` `end_defun`

This command moves point forward to the closing parenthesis that corresponds to the last opening parenthesis against the left margin.

▶ `Esc Ctrl R` `move_defun_to_screen_top`

This command moves the form beginning with the last opening parenthesis against the left margin to the top of the screen.

► `[Esc] [H]`

`mark_defun`

This command marks the region between the last opening parenthesis against the left margin and the closing parenthesis that corresponds to it and moves point just to the right of this closing parenthesis. This is the easiest way to mark a function before sending it to the Listener buffer for evaluation (see below). For larger pieces of code, you can also use the standard way of marking a region with `[Ctrl @]` and point.

## Common LISP Mode

Common LISP mode is one of the three LISP-related language modes provided by EMACS. It is designed for use with PRIME Common LISP. Common LISP mode contains all of LISP mode's editing and formatting functionality, as well as the ability to send all or selected portions of your source file to the Listener buffer for interpretation or compilation by PRIME Common LISP.

### Entering Common LISP Mode

The following command turns on PRIME Common LISP mode in the current buffer:

► `cl_on`

This command turns on both Common LISP and LISP modes in the current buffer. Once Common LISP mode is invoked, `common_lisp`, `lisp` appears on the EMACS status line.

### Exiting From Common LISP Mode

The following command turns off PRIME Common LISP mode in the current buffer:

► `cl_off`

### Additional PRIME Common LISP Mode Information

In addition to the LISP mode functionality described above, the following keybindings and extended commands are effective only in Common LISP mode.

► `)`

`cl_close_paren`

This command moves point briefly to the opening parenthesis that corresponds to the closing parenthesis just typed. After a pause, EMACS returns point to the position immediately following the closing parenthesis just typed. The pause at the opening parenthesis usually lasts 750 milliseconds. If you want to change the default duration of the pause, set the EMACS variable `lisp.paren_time` to the new value. A value of 0 turns off `cl_close_paren`.

▶ `Return` `cl_cr`

This command pretty prints your code as you type it. Pretty printing arranges the code so that indentation is consistent and the structure of expressions is clear.

▶ `cl_listener`

This command replaces the current buffer with a buffer named [listener]; **listener, lisp** appears on the EMACS status line. In the listener buffer, you have a right angle-bracket (>) prompt and can enter commands at PRIME Common LISP top level (not EMACS Common LISP mode). You can split the screen so that your source text appears in one window and the Listener buffer in the other. Your entire source file, or selected portions of it, can be passed to the Listener buffer for interpretation (evaluation) or compilation by PRIME Common LISP. The Listener buffer can be edited and saved in a file as a record of your LISP session.

▶ `Esc E` `cl_send_region`

This command sends the current region to the Listener buffer for interpretation (evaluation) by PRIME Common LISP. After you issue this command, **working** appears in the EMACS minibuffer area. If you are using only one window, EMACS splits the screen; your source code is in the top window, and the Listener buffer appears in the bottom window. The cursor moves to the bottom window, and the current region is evaluated there by PRIME Common LISP. The right angle-bracket (>) prompt confirms that you can enter commands at PRIME Common LISP top level.

▶ `Esc Z` `cl_compile_region`

This command sends the current region to the Listener buffer for compilation by PRIME Common LISP. The screen behavior associated with the Listener buffer is the same as that described under the function `cl_send_region`.

▶ `Esc M` `cl_macroexpand_region`

This command sends the current region to the Listener buffer for macro expansion by PRIME Common LISP. The screen behavior associated with the Listener buffer is the same as that described under the function `cl_send_region`.

▶ `Esc T` `cl_send_file`

This command saves the file in the current buffer and sends it to the Listener buffer for evaluation by PRIME Common LISP. An acknowledgement message shows the pathname of the loaded file. The screen behavior associated with the Listener buffer is the same as that described under the function `cl_send_region`.

▶ 

Esc	=
-----	---

**cl\_describe\_current\_symbol**

This command uses Common LISP's **describe** function to print information about the current symbol.

▶ 

Ctrl X	?
--------	---

**cl\_help**

This command displays a summary of Common LISP mode's special keybindings and their meanings.

## Listener Mode

Listener mode is one of the three LISP-related language modes provided by EMACS. Listener mode is usable only in the Listener buffer. It gives you direct access to PRIME Common LISP without leaving EMACS. You can split the screen so that your source text appears in one window and the Listener buffer in the other. The Listener buffer can be edited and saved in a file as a record of your LISP session.

### Entering Listener Mode

To enter the Listener buffer and invoke Listener mode from EMACS, issue the **cl\_listener** command, described in the section Common LISP Mode above. Listener mode can be invoked only in a special buffer named [listener]. Once Listener mode is invoked, **listener**, **lisp** appears on the EMACS status line. In the Listener buffer, the right-angle bracket (>) prompt confirms that you can enter commands at PRIME Common LISP top level.

### Exiting from Listener Mode

You can change back and forth from the Listener buffer to other buffers with the standard EMACS commands. You may exit from EMACS when you are in the Listener buffer.

### Additional Listener Mode Information

Listener mode includes all of LISP mode's functionality. However, because a function's opening parenthesis is no longer against the left margin in the Listener buffer, the following LISP mode keybindings and extended commands, all relating to cursor movement and region marking, are not recommended for use in Listener mode:

<table border="1"><tr><td>Esc</td><td>Ctrl A</td></tr></table>	Esc	Ctrl A	<b>begin_defun</b>
Esc	Ctrl A		
<table border="1"><tr><td>Esc</td><td>Ctrl E</td></tr></table>	Esc	Ctrl E	<b>end_defun</b>
Esc	Ctrl E		
<table border="1"><tr><td>Esc</td><td>Ctrl R</td></tr></table>	Esc	Ctrl R	<b>move_defun_to_screen_top</b>
Esc	Ctrl R		
<table border="1"><tr><td>Esc</td><td>Ctrl H</td></tr></table>	Esc	Ctrl H	<b>mark_defun</b>
Esc	Ctrl H		

The following keybindings and extended commands from Common LISP mode are available in Listener mode:

<code>Esc</code>	<code>E</code>	<code>cl_send_region</code>
<code>Esc</code>	<code>Z</code>	<code>cl_compile_region</code>
<code>Esc</code>	<code>M</code>	<code>cl_macroexpand_region</code>
<code>Esc</code>	<code>=</code>	<code>cl_describe_current_symbol</code>
<code>)</code>		<code>cl_close_paren</code>

In addition, Listener mode provides the following special keybindings and extended commands:

▶ `Return` `cl_send_lisp_line`

This command sends blocked input to PRIME Common LISP for evaluation. EMACS maintains a pointer to the start of an area of blocked input. A carriage return sends the blocked input and updates the pointer. Input is sent line by line and can be edited only before it is sent. If you accidentally corrupt the pointer, EMACS resets it and displays an error message.

▶ `Ctrl X ?` `cl_listener_help`

This command displays a summary of Listener mode's special key bindings and their meanings.

## Additional Information About Compiling and Debugging Programs

If you are writing functions in PEEL, Prime's EMACS Extension Language, and have not invoked a language mode, you can compile your functions from EMACS using the `PI` function. (See the *EMACS Extension Writing Guide* for more information.)

You can compile or interpret all or part of a Common LISP program from Common LISP mode. See the previous description of Command LISP mode for details.

If you are creating COBOL, C, FORTRAN, or RPG programs, you can compile the program from EMACS with the compile command described below.

▶ `compile`

This command compiles the source code in the current buffer if you have invoked a language mode in this buffer. To include the compiler options on the compile command line, you must type them in uppercase letters. An example of a compile command line for COBOL mode that shows the compiler options on the command line is shown below.

```
compile -LISTING -DEBUG
```

After you issue this command, EMACS checks to make sure that you have saved the current buffer text. If you have not, EMACS saves it for you. If there is more than one window on the screen, EMACS removes the other windows. It then tells PRIMOS to compile the current buffer as a phantom process. After the phantom executes, EMACS displays the message produced by the language compiler in the minibuffer. For example, for the CBL compiler, this message might be:

```
No errors in compilation
```

## Compile Command Variables

The compile command has a number of internal variables that let you control how it works.

▶ **lm\_list\_path***\$ string*

This string variable defines the directory to which the compiler should write the listing file. It always has a value that is set either by the user or by default. If you do not set this variable, EMACS writes the file to your current directory.

▶ **lm\_bin\_path***\$ string*

This string variable defines the directory to which the compiler should write the binary (object) file. If you do not set this variable, EMACS writes the file to your current directory.

▶ **lm\_max\_num\_errors***\$ integer*

This integer variable specifies the maximum number of compiler error diagnostics and warnings that EMACS should issue. The default is 100, which is the maximum acceptable value.

Every diagnostic adds time to the execution of the compile command. Therefore, assigning a smaller value to this variable causes faster compilation.

▶ **lm\_max\_error\_size***\$ integer*

This integer variable tells EMACS what the maximum size of the error window should be. Smaller error messages are placed in smaller windows. If the error message is larger than the maximum window size of 10 lines, EMACS displays only the top part of the message.

▶ **lm\_forward\_error\_key***\$ keypath*

This string variable indicates what keypath should be used to invoke **lm\_next\_error**\$. The default is `Ctrl X N`.

► `lm_prev_error_key$ keypath`

This string variable indicates what keypath should be used to invoke `lm_prev_error$`. The default is `Ctrl X P`.

The following string variables define which compiler options should be used when the program is compiled. Prime supports all regular compiler options for each language. If a variable is not set to a value, no options are used.

► `cbl_compile_options$ options`

► `cc_compile_options$ options`

► `ftn_compile_options$ options`

► `f77_compile_options$ options`

► `vrpg_compile_options$ options`

## Initializing the Variables

You can initialize the compile command variables in two ways. The next section discusses each method.

- Make them part of a library file.
- Set them manually with the `Esc X` command.

You can initialize the variables in a library file so that you do not have to reset them for each EMACS session. Use the `setq` extension language statement. For example, to initialize the `lm_max_num_errors$` integer variable to 40, you would include the following statement in your library file:

```
(setq lm_max_num_errors$ 40)
```

Similarly, to initialize the `lm_bin_path$` string variable to the value `"*>bin"`, use this statement:

```
(setq lm_bin_path$ ">bin")
```

An example of how to set CBL compiler options for listing and debug in a library file is shown below:

```
(setq cbl_compile_options$ "-LISTING -DEBUG")
```

### Note

The variable name must be in *lowercase*, and the compiler options must be in *uppercase* letters.

To set these variables manually, you can use the following commands. EMACS issues appropriate prompts. Again, be sure to type the compiler options in uppercase.

```
set_lm_list_path$
set_lm_bin_path$
set_cbl_compile$
set_cc_compile$
set_ftn_compile$
set_f77_compile$
set_vrpg_compile$
set_max_num_errors$
set_lm_max_error_size$
```

## Debugging Programs

When a language mode is in effect and you have issued the compile command, EMACS provides a message at the bottom of the screen, indicating the number of errors, the command to view the next error, and the command to view the previous error. Either of the following commands divides the screen and displays one or more errors in the program. The specific commands for viewing the error(s) are:

▶ Ctrl X N lm\_next\_error\$

This command divides the screen and moves the cursor to the next error or warning in the source code. It prints the error or warning in the top screen buffer. You may alter the code as it appears in the lower screen and recompile. If you type this command after the last error message, EMACS informs you and displays only the source code. To change the bindings of this command, use the **lm\_forward\_error\_key\$** command discussed above.

▶ Ctrl X P lm\_prev\_error\$

This command divides the screen and moves the cursor to the previous error or warning in the source code. It prints the error or warning in the top screen buffer. You may alter the code as it appears in the lower screen and recompile the program. If you type this command when the cursor is placed before the first error message EMACS tells you this and displays only the source code. To change the bindings of this command, use the **lm\_prev\_error\_key\$** command discussed above.

---

# Appendices

---

# A

## Alphabetical Summary of Commands

This appendix lists all of the EMACS commands alphabetically by keybinding. Each command is followed by its corresponding function name and a brief description.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
Ctrl _	help_on_tap	Accesses help function.
Ctrl @	mark	Sets mark at cursor.
Ctrl A	begin_line	Moves cursor to beginning of line.
Ctrl B	back_char	Moves cursor back one character.
Ctrl C	reexecute	Reexecutes most recent command.
Ctrl D	delete_char	Deletes character after point.
Ctrl E	end_line	Moves cursor to end of line.
Ctrl F	forward_char	Moves cursor forward one character.
Ctrl G	abort_command	Aborts most recent command.
Ctrl G	abort_minibuffer	Aborts minibuffer when in minibuffer mode.
Ctrl H	rubout_char	Deletes preceding character.
Ctrl I	type_tab	Moves cursor to next tab stop.
Ctrl K	kill_line	Kills current line.
Ctrl L	refresh	Refreshes screen.
Return	cr	Inserts carriage return just before current cursor.
Return	wrap	Inserts carriage return in fill mode.
Return	exit_minibuffer	Exits minibuffer when in minibuffer mode.
Ctrl N	next_line_command	Moves cursor to next line.
Ctrl O	open_line	Opens line without moving the cursor.
Ctrl P	break	Prompts before breaking to PRIMOS command level.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
Ctrl Q	^q_quote_command	Inserts a nonalphabetic character into your text.
Ctrl R	reverse_search_command	Searches backward.
Ctrl S	^s_forward_search_command	Searches forward (stops output on some terminals).
Ctrl T	twiddle	Transposes two characters preceding cursor.
Ctrl U	multiplier	Multiplies prefix count.
Ctrl U Ctrl @		Moves cursor back one position of mark on ring of marks.
Ctrl U Ctrl X Ctrl V	view_off	Turns view mode off.
Ctrl V	next_page	Scrolls screen display forward 18 lines.
Ctrl W	kill_region	Moves region between mark and cursor to kill ring.
Ctrl X Ctrl B	list_buffers	Lists all active buffers.
Ctrl X Ctrl C	quit	Leaves EMACS.
Ctrl X Ctrl E	primos_command	Executes a PRIMOS command.
Ctrl X Ctrl F	find_file	Finds a file and reads it into buffer named filename.
Ctrl X Ctrl G	ignore_prefix	Aborts the command.
Ctrl X Ctrl H	backward_kill_sentence	Kills a sentence from cursor back to beginning.
Ctrl X Ctrl I	insert_tab	Moves cursor and text to next tab position.
Ctrl X Ctrl K	backward_kill_line	Kills from cursor to beginning of line.
Ctrl X Ctrl L	lowercase_region	Converts region to lowercase.
Ctrl X Return	cret_indent_relative	Moves cursor to new line that is indented the same as previous line.
Ctrl X Ctrl O	delete_blank_lines	Deletes blank lines immediately above and below cursor.
Ctrl X Ctrl R	read_file	Reads a file into current buffer.
Ctrl X Ctrl S	save_file	Saves a file.
Ctrl X Ctrl T	toggle_redisplay	Toggles the redisplay mode.
Ctrl X Ctrl U	uppercase_region	Converts region to uppercase.
Ctrl X Ctrl V	view_file	Lets a user look at a file.
Ctrl X Ctrl W	mod_write_file	Prompts you before writing buffer to file.
Ctrl X Ctrl X	exchange_mark	Exchanges mark and cursor.
Ctrl X Ctrl Z Ctrl A	backward_clause	Moves cursor backward one clause.
Ctrl X Ctrl Z Ctrl E	forward_clause	Moves point forward one clause.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
Ctrl X Ctrl Z Ctrl F	get_filename	Displays pathname of current buffer.
Ctrl X Ctrl Z Ctrl G	ignore_prefix	Aborts most recent command.
Ctrl X Ctrl Z Ctrl H	backward_kill_clause	Kills backward one clause.
Ctrl X Ctrl Z Ctrl K	forward_kill_clause	Kills forward one clause.
Ctrl X Ctrl Z Ctrl Y	yank_kill_text	Inserts text saved by view_kill_ring.
Ctrl X Ctrl Z <	mark_top	Marks top of buffer.
Ctrl X Ctrl Z >	mark_bottom	Marks bottom of buffer.
Ctrl X Ctrl Z A	append_to_file	Appends current region to a file.
Ctrl X Ctrl Z F	take_right_margin	Sets right margin from current cursor position. Column 10 is cutoff point.
Ctrl X Ctrl Z I	insert_buf	Inserts specified buffer at cursor.
Ctrl X Ctrl Z K	view_kill_ring	Displays contents of kill ring.
Ctrl X Ctrl Z P	prepend_to_file	Prepends current region to a file.
Ctrl X Ctrl Z S	center_line	Centers current line.
Ctrl X Ctrl Z Ctrl V	view_lines	Views lines toggled off with toggle_redisplay.
Ctrl X (	collect_macro	Starts collecting macro.
Ctrl X )	finish_macro	Stops collecting macro.
Ctrl X +	spd_add_region	Defines a region as expansion of speed-type abbreviation.
Ctrl X .	take_left_margin	Sets left fill margin to column containing the cursor.
Ctrl X 1	mod_one_window	Transforms a split screen into one.
Ctrl X 2	mod_split_window	Divides screen horizontally into two screens.
Ctrl X 3	mod_split_window_stay	Splits screen; keeps cursor in current one.
Ctrl X 4	select_any_window	Cycles cursor through all windows.
Ctrl X <	back_place_holder	Moves to previous speed-type place holder.
Ctrl X =	tell_position	Gives line and cursor position.
Ctrl X >	forward_place_holder	Moves to next speed-type placeholder.
Ctrl X A	append_to_buf	Appends current region to a buffer.
Ctrl X B	mod_select_buf	Moves you to the specified buffer.
Ctrl X B Return		Returns you to previous buffer.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<b>Ctrl X</b> <b>D</b>	explore	Allows user to explore a directory.
<b>Ctrl X</b> <b>E</b>	execute_macro	Executes current keyboard macro.
<b>Ctrl X</b> <b>F</b>	set_right_margin	Sets right margin to specified value.
<b>Ctrl X</b> <b>H</b>	mark_whole	Marks whole buffer as region.
<b>Ctrl X</b> <b>I</b>	insert_file	Inserts file into buffer at cursor.
<b>Ctrl X</b> <b>O</b>	other_window	Moves cursor between current window and previous window.
<b>Ctrl X</b> <b>P</b>	prepend_to_buf	Prepends current region to a buffer.
<b>Ctrl X</b> <b>Q</b>	quote_command	Inserts a nonalphabetic character into your text.
<b>Ctrl X</b> <b>R</b>	repaint	Moves cursor to first column of first line on screen.
<b>Ctrl X</b> <b>S</b>	save_file	Saves file.
<b>Ctrl X</b> <b>V</b>	scroll_other_backward	Scrolls other window backward.
<b>Ctrl X</b> <b>[</b>	backward_para	Moves cursor back to the beginning of the paragraph.
<b>Ctrl X</b> <b>]</b>	forward_para	Moves cursor forward to the end of the paragraph.
<b>Ctrl X</b> <b>-</b>	spd_unexpand	Removes speed-type expansion from current buffer.
<b>Ctrl X</b> <b>{</b>	horiz_left	Shifts the current window left 40 spaces.
<b>Ctrl X</b> <b>}</b>	horiz_right	Shifts the current window right 40 spaces.
<b>Ctrl Y</b>	yank_region	Restores the last text deleted.
<b>Ctrl Z</b>	prev_line_command	Moves cursor up one line.
<b>Esc</b> <b>Ctrl D</b>	kill_rest_of_buffer	Kills from cursor to end of buffer.
<b>Esc</b> <b>Ctrl G</b>	ignore_prefix	Aborts most recent command.
<b>Esc</b> <b>Ctrl H</b>	rubout_word	Deletes word before cursor.
<b>Esc</b> <b>Ctrl I</b>	indent_to_fill_prefix	Moves line to left margin.
<b>Esc</b> <b>Ctrl O</b>	split_line	Breaks line at point and indents next line at same column.
<b>Esc</b> <b>Ctrl V</b>	scroll_other_forward	Scrolls other window forward.
<b>Esc</b> <b>Ctrl Y</b>	yank_minibuffer	Inserts the response to the previous minibuffer prompt into your current buffer.
<b>Esc</b> <b>Esc</b>	pl_minibuffer	Enables PL minibuffer.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<b>Esc</b> <b>SPACE</b>	leave_one_white	Deletes all but one space around cursor.
<b>Esc</b> <b>%</b>	query_replace	Executes query_replace function.
<b>Esc</b> <b>-</b>	esc_minus	Prefix minus.
<b>Esc</b> <b>0</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>1</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>2</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>3</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>4</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>5</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>6</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>7</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>8</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>9</b>	esc_digit	Prefix digit.
<b>Esc</b> <b>&lt;</b>	move_top	Moves cursor to top of buffer.
<b>Esc</b> <b>&gt;</b>	move_bottom	Moves cursor to bottom of buffer.
<b>Esc</b> <b>?</b>	explain_key	Alternate method of invoking the C help option.
<b>Esc</b> <b>@</b>	mark_end_of_word	Marks end of current word.
<b>Esc</b> <b>A</b>	backward_sentence	Moves cursor to the beginning of the sentence.
<b>Esc</b> <b>B</b>	back_word	Moves cursor back to the beginning of a word.
<b>Esc</b> <b>C</b>	capinitial	Capitalizes current word.
<b>Esc</b> <b>D</b>	delete_word	Deletes a word.
<b>Esc</b> <b>E</b>	forward_sentence	Moves cursor to the end of the sentence.
<b>Esc</b> <b>F</b>	forward_word	Moves cursor forward one word.
<b>Esc</b> <b>G</b>	goto_line	Moves cursor to specified line.
<b>Esc</b> <b>H</b>	mark_para	Marks a paragraph.
<b>Esc</b> <b>I</b>	indent_relative	Indents current line relative to previous one.
<b>Esc</b> <b>K</b>	forward_kill_sentence	Kills a sentence forward from point.
<b>Esc</b> <b>L</b>	lowercase_word	Converts word to lowercase.
<b>Esc</b> <b>M</b>	back_to_nonwhite	Moves cursor to first nonblank character on line.
<b>Esc</b> <b>N</b>	next_buf	Replaces current buffer with the next buffer.
<b>Esc</b> <b>P</b>	prev_buf	Replaces current buffer with the previous buffer.
<b>Esc</b> <b>Q</b>	fill_para	Fills paragraph according to set columns.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<b>Esc</b> <b>R</b>	reverse_search_command	Searches backward for specified string.
<b>Esc</b> <b>S</b>	forward_search_command	Searches forward for specified string.
<b>Esc</b> <b>T</b>	transpose_word	Transposes two words before and after cursor.
<b>Esc</b> <b>U</b>	uppercase_word	Converts word to uppercase.
<b>Esc</b> <b>V</b>	back_page	Scrolls screen display back 18 lines.
<b>Esc</b> <b>W</b>	copy_region	Copies region to kill ring.
<b>Esc</b> <b>X</b>	extend_command	Extended command prefix.
<b>Esc</b> <b>Y</b>	yank_replace	Yanks previous region from kill ring and replaces previous yank.
<b>Esc</b> <b>\</b>	white_delete	Deletes space around cursor.
<b>Esc</b> <b>~</b>	unmodify	Unmodifies buffer.
<b>Esc</b> <b>^</b>	merge_lines	Merges two lines together.
<b>Extended Commands</b>		
<b>Esc</b> <b>X</b>	#	Tells if line numbering is in effect.
<b>Esc</b> <b>X</b>	#off	Turns off line numbering.
<b>Esc</b> <b>X</b>	#on	Turns on line numbering.
<b>Esc</b> <b>X</b>	2d	Tells if 2d is in effect.
<b>Esc</b> <b>X</b>	2doff	Turns off two-dimensional mode.
<b>Esc</b> <b>X</b>	2don	Turns on two-dimensional mode.
<b>Esc</b> <b>X</b>	all_modes_off	Turns off all modes.
<b>Esc</b> <b>X</b>	apropos	Alternate method of invoking the A help option.
<b>Esc</b> <b>X</b>	back_tab	Moves cursor to previous tab stop.
<b>Esc</b> <b>X</b>	case?	Tells if case matching occurs during search function.
<b>Esc</b> <b>X</b>	case_off	Causes EMACS not to distinguish between cases of letters.
<b>Esc</b> <b>X</b>	case_on	Causes EMACS to distinguish between cases of letters.
<b>Esc</b> <b>X</b>	case_replace_off	Treats uppercase and lowercase letters the same when replacing.
<b>Esc</b> <b>X</b>	case_replace_on	Distinguishes uppercase letters from lowercase when replacing.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<input type="checkbox"/> Esc <input type="checkbox"/> X	case_replace?	Tells whether cases are distinguished when replacing.
<input type="checkbox"/> Esc <input type="checkbox"/> X	date	Inserts the current day, month, year: MON, 28 DEC 1987
<input type="checkbox"/> Esc <input type="checkbox"/> X	default_tabs	Sets up default tabs every five spaces.
<input type="checkbox"/> Esc <input type="checkbox"/> X	delete_buffer	Deletes current buffer; does not put on kill ring.
<input type="checkbox"/> Esc <input type="checkbox"/> X	delete_region	Deletes region without placing on kill ring.
<input type="checkbox"/> Esc <input type="checkbox"/> X	describe	Alternate method of invoking the D help option.
<input type="checkbox"/> Esc <input type="checkbox"/> X	display_buffer	Displays information about current buffer.
<input type="checkbox"/> Esc <input type="checkbox"/> X	display_debug	Displays information about debug.
<input type="checkbox"/> Esc <input type="checkbox"/> X	display_terminal	Displays information about your terminal.
<input type="checkbox"/> Esc <input type="checkbox"/> X	display_window	Displays information about current window.
<input type="checkbox"/> Esc <input type="checkbox"/> X	dt	Inserts the current date and time: 12/29/87 09:59:37
<input type="checkbox"/> Esc <input type="checkbox"/> X	dump_file	Partially compiles PEEL file in current buffer to a fadump file with .EFASL suffix.
<input type="checkbox"/> Esc <input type="checkbox"/> X	europe_dt	Inserts the current date in the European way: 29/12/87
<input type="checkbox"/> Esc <input type="checkbox"/> X	expand_macro	Expands keyboard macro into PEEL source code.
<input type="checkbox"/> Esc <input type="checkbox"/> X	explain_key	Alternate method of invoking the C help option.
<input type="checkbox"/> Esc <input type="checkbox"/> X	fill_off	Turns off fill mode.
<input type="checkbox"/> Esc <input type="checkbox"/> X	fill_on	Turns on fill mode in current buffer.
<input type="checkbox"/> Esc <input type="checkbox"/> X	forward_search_again	Searches forward again.
<input type="checkbox"/> Esc <input type="checkbox"/> X	fundamental	Turns on fundamental mode.
<input type="checkbox"/> Esc <input type="checkbox"/> X	get_bufname	Inserts current buffer name in buffer at cursor.
<input type="checkbox"/> Esc <input type="checkbox"/> X	get_tab	Restores previously saved tabs.
<input type="checkbox"/> Esc <input type="checkbox"/> X	global_tabs	Activates default tabs.
<input type="checkbox"/> Esc <input type="checkbox"/> X	hcol	Sets or checks horizontal column.
<input type="checkbox"/> Esc <input type="checkbox"/> X	help_on_tap	Invokes help facility and lists command options.
<input type="checkbox"/> Esc <input type="checkbox"/> X	hscroll	Uses current column position to set hcol.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<code>Esc</code> <code>X</code>	<code>insert_version</code>	Inserts version number of EMACS into buffer.
<code>Esc</code> <code>X</code>	<code>load_compiled</code>	Loads a fasdump file saved with <code>dump_file</code> command.
<code>Esc</code> <code>X</code>	<code>load_pl_source</code>	Compiles and executes the source code in a PL source file.
<code>Esc</code> <code>X</code>	<code>local_tabs</code>	Activates local (to buffer) tabs.
<code>Esc</code> <code>X</code>	<code>new_features</code>	Displays the file found in <code>EMACS*&gt;INFO&gt;NEW_FEATURES_INFO</code> .
<code>Esc</code> <code>X</code>	<code>one_window</code>	Transforms a split screen into one. (Similar to <code>mod_one_window</code> .)
<code>Esc</code> <code>X</code>	<code>overlay_off</code>	Turns off overlay mode.
<code>Esc</code> <code>X</code>	<code>overlay_on</code>	Turns on overlay mode.
<code>Esc</code> <code>X</code>	<code>overlay_rubout</code>	In overlay mode, erases previous character.
<code>Esc</code> <code>X</code>	<code>pl</code>	Compiles and executes contents of current buffer.
<code>Esc</code> <code>X</code>	<code>popmark</code>	Pops mark off mark stack.
<code>Esc</code> <code>X</code>	<code>pushmark</code>	Sets mark, pushes previous one onto mark stack.
<code>Esc</code> <code>X</code>	<code>quote_command</code>	Identical to <code>^q_quote_command</code> .
<code>Esc</code> <code>X</code>	<code>reject</code>	Invalid command.
<code>Esc</code> <code>X</code>	<code>replace</code>	Replaces one string with another globally.
<code>Esc</code> <code>X</code>	<code>reset</code>	Resets windows and columns.
<code>Esc</code> <code>X</code>	<code>reverse_search_again</code>	Searches backward again.
<code>Esc</code> <code>X</code>	<code>save_all_files</code>	Saves all files you modify.
<code>Esc</code> <code>X</code>	<code>save_tab</code>	Saves current tab positions in a file.
<code>Esc</code> <code>X</code>	<code>select_buf</code>	Moves you to the specified buffer. (Similar to <code>mod_select_buf</code> .)
<code>Esc</code> <code>X</code>	<code>set_hscroll</code>	Prompts you for the <code>hcol</code> value used in horizontal scrolling.
<code>Esc</code> <code>X</code>	<code>set_key</code>	Binds function to a key on a per-buffer basis.
<code>Esc</code> <code>X</code>	<code>set_left_margin</code>	Sets left margin to position of cursor.
<code>Esc</code> <code>X</code>	<code>set_mode</code>	Sets buffer to specified mode.
<code>Esc</code> <code>X</code>	<code>set_mode_key</code>	Binds function to a key on a per-mode basis.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
 	set_permanent_key	Binds function to a key for an entire session.
 	setmark	Sets mark at current cursor.
 	settab	Sets tabs to specified values.
 	set_tab	Sets tabs to specified values.
 	set_tabs	Same as settab and set_tab.
 	settabs_from_table	Sets tabs based on column position of words.
 	setft	Same as settabs_from_table.
 	set_user_type	Sets user type for file hook procedure.
 	sort_dt	Inserts year, day, month: 87/12/29
 	split_window	Divides screen horizontally into two screens. (Similar to mod_split_window.)
 	split_window_stay	Splits screen; keeps cursor in current one. (Similar to mod_split_window_stay.)
 	spd_add	Adds speed_type abbreviation.
 	spd_add_modal	Adds abbreviation for specific mode.
 	spd_compile	Compiles current buffer into speed-type file.
 	spd_delete	Deletes speed-type abbreviation.
 	spd_list	Gives information about specific symbol.
 	spd_list_all	Gives information about all speed-type abbreviations.
 	spd_list_file	Lists abbreviations for specific file.
 	spd_load_file	Loads a speed-type abbreviation file.
 	spd_off	Turns off speed-type abbreviations.
 	spd_on	Turns on speed-type abbreviations.
 	spd_save_file	Saves changes made to current speed-type environment.
 	tablist	Sets tabs from a numbered list.
 	tell_left_margin	Gives current setting of left margin.
 	tell_modes	Displays all modes for buffer.
 	tell_position	Displays line and cursor position.

<i>Command</i>	<i>Function Name</i>	<i>Description</i>
<input type="checkbox"/> Esc <input type="checkbox"/> X	tell_right_margin	Gives current setting of right margin.
<input type="checkbox"/> Esc <input type="checkbox"/> X	trim_date	Inserts day, month, year: 29 Dec 1987
<input type="checkbox"/> Esc <input type="checkbox"/> X	trim_dt	Inserts month, day, year: 12/29/87
<input type="checkbox"/> Esc <input type="checkbox"/> X	type_tab	Moves cursor to next tab stop.
<input type="checkbox"/> Esc <input type="checkbox"/> X	untidy	Unjustifies a paragraph.
<input type="checkbox"/> Esc <input type="checkbox"/> X	view	Same as view_file except that bound function keys can be used.
<input type="checkbox"/> Esc <input type="checkbox"/> X	view_on	Turns view mode on.
<input type="checkbox"/> Esc <input type="checkbox"/> X	wallpaper	Lists all EMACS commands and functions.
<input type="checkbox"/> Esc <input type="checkbox"/> X	which_tabs	Tells which tabs are in use.
<input type="checkbox"/> Esc <input type="checkbox"/> X	write_file	Writes specified file.
<input type="checkbox"/> Esc <input type="checkbox"/> X	vsplit	Splits window vertically at point.

---

## B

# Prime Extended Character Set

As of Rev. 21.0, Prime has expanded its character set from 128 to 256 characters. The basic character set remains the same as it was before Rev. 21.0: it is the ASCII 7-bit set (called ASCII-7), with the eighth bit turned on. However, the eighth bit is now significant; when it is turned off, it signifies a different character. This expanded character set is called the Prime Extended Character Set (Prime ECS).

The pre-Rev. 21.0 character set is a proper subset of Prime ECS. Software written before Rev. 21.0 will continue to run exactly as it did before. Software written at Rev. 21.0 that does not use the new characters needs no special coding to use the old ones.

Prime ECS support is automatic at Rev. 21.0, when you may begin to use characters that have the eighth bit turned off. However, be aware that the extra characters are not available on most printers and terminals. A terminal supports Prime ECS if

- It uses ASCII-8 as its internal character set, and
- The TTY8 protocol is configured on your asynchronous line.

If you do not know whether your terminal supports Prime ECS, ask your System Administrator.

Table B-1 shows the Prime Extended Character Set. The pre-Rev. 21.0 character set consists of the characters with decimal values 128 through 255 (octal values 200 through 377). The characters added at Rev. 21 all have decimal values less than 128 (octal values less than 200).

## Specifying Prime ECS Characters

### Direct Entry

On terminals that support Prime ECS, you can enter the characters directly. For information on how to do this, see the appropriate manual for your terminal. EMACS displays ECS characters as question marks or rectangular blocks. You can check their octal values with `Ctrl X` = (the `tell_position` command).

**Table B-1**  
**Prime Extended Character Set**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	RES1	Reserved for future standardization	0000 0000	000	00	000
	RES2	Reserved for future standardization	0000 0001	001	01	001
	RES3	Reserved for future standardization	0000 0010	002	02	002
	RES4	Reserved for future standardization	0000 0011	003	03	003
	IND	Index	0000 0100	004	04	004
	NEL	Next line	0000 0101	005	05	005
	SSA	Start of selected area	0000 0110	006	06	006
	ESA	End of selected area	0000 0111	007	07	007
	HTS	Horizontal tabulation set	0000 1000	008	08	010
	HTJ	Horizontal tab with justify	0000 1001	009	09	011
	VTS	Vertical tabulation set	0000 1010	010	0A	012
	PLD	Partial line down	0000 1011	011	0B	013
	PLU	Partial line up	0000 1100	012	0C	014
	RI	Reverse index	0000 1101	013	0D	015
	SS2	Single shift 2	0000 1110	014	0E	016
	SS3	Single shift 3	0000 1111	015	0F	017
	DCS	Device control string	0001 0000	016	10	020
	PU1	Private use 1	0001 0001	017	11	021
	PU2	Private use 2	0001 0010	018	12	022
	STS	Set transmission state	0001 0011	019	13	023
	CCH	Cancel character	0001 0100	020	14	024
	MW	Message waiting	0001 0101	021	15	025
	SPA	Start of protected area	0001 0110	022	16	026
	EPA	End of protected area	0001 0111	023	17	027
	RES5	Reserved for future standardization	0001 1000	024	18	030
	RES6	Reserved for future standardization	0001 1001	025	19	031
	RES7	Reserved for future standardization	0001 1010	026	1A	032
	CSI	Control sequence introducer	0001 1011	027	1B	033
	ST	String terminator	0001 1100	028	1C	034
	OSC	Operating system command	0001 1101	029	1D	035
	PM	Privacy message	0001 1110	030	1E	036

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	APC	Application program command	0001 1111	031	1F	037
	NBSP	No-break space	0010 0000	032	20	040
¡	INVE	Inverted exclamation mark	0010 0001	033	21	041
¢	CENT	Cent sign	0010 0010	034	22	042
£	PND	Pound sign	0010 0011	035	23	043
¤	CURR	Currency sign	0010 0100	036	24	044
¥	YEN	Yen sign	0010 0101	037	25	045
¦	BBAR	Broken bar	0010 0110	038	26	046
§	SECT	Section sign	0010 0111	039	27	047
¨	DIA	Diaeresis, umlaut	0010 1000	040	28	050
©	COPY	Copyright sign	0010 1001	041	29	051
ª	FOI	Feminine ordinal indicator	0010 1010	042	2A	052
«	LAQM	Left angle quotation mark	0010 1011	043	2B	053
¬	NOT	Not sign	0010 1100	044	2C	054
	SHY	Soft hyphen	0010 1101	045	2D	055
®	TM	Registered trademark sign	0010 1110	046	2E	056
	MACN	Macron	0010 1111	047	2F	057
°	DEGR	Degree sign	0011 0000	048	30	060
±	PLMI	Plus/minus sign	0011 0001	049	31	061
²	SPS2	Superscript two	0011 0010	050	32	062
³	SPS3	Superscript three	0011 0011	051	33	063
´	AAC	Acute accent	0011 0100	052	34	064
µ	LCMU	Lowercase Greek letter µ, micro sign	0011 0101	053	35	065
¶	PARA	Paragraph sign, Pilgrow sign	0011 0110	054	36	066
•	MIDD	Middle dot	0011 0111	055	37	067
¸	CED	Cedilla	0011 1000	056	38	070
¹	SPS1	Superscript one	0011 1001	057	39	071
º	MOI	Masculine ordinal indicator	0011 1010	058	3A	072
»	RAQM	Right angle quotation mark	0011 1011	059	3B	073
¼	FR14	Common fraction one-quarter	0011 1100	060	3C	074

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
½	FR12	Common fraction one-half	0011 1101	061	3D	075
¾	FR34	Common fraction three-quarters	0011 1110	062	3E	076
¿	INVQ	Inverted question mark	0011 1111	063	3F	077
À	UCAG	Uppercase A with grave accent	0100 0000	064	40	100
Á	UCAA	Uppercase A with acute accent	0100 0001	065	41	101
Â	UCAC	Uppercase A with circumflex	0100 0010	066	42	102
Ã	UCAT	Uppercase A with tilde	0100 0011	067	43	103
Ä	UCAD	Uppercase A with diaeresis	0100 0100	068	44	104
Å	UCAR	Uppercase A with ring above	0100 0101	069	45	105
Æ	UCAE	Uppercase diphthong Æ	0100 0110	070	46	106
Ç	UCCC	Uppercase C with cedilla	0100 0111	071	47	107
È	UCEG	Uppercase E with grave accent	0100 1000	072	48	110
É	UCEA	Uppercase E with acute accent	0100 1001	073	49	111
Ê	UCEC	Uppercase E with circumflex	0100 1010	074	4A	112
Ë	UCED	Uppercase E with diaeresis	0100 1011	075	4B	113
Ì	UCIG	Uppercase I with grave accent	0100 1100	076	4C	114
Í	UCIA	Uppercase I with acute accent	0100 1101	077	4D	115
Î	UCIC	Uppercase I with circumflex	0100 1110	078	4E	116
Ï	UCID	Uppercase I with diaeresis	0100 1111	079	4F	117
Ð	UETH	Uppercase Icelandic letter <u>Eth</u>	0101 0000	080	50	120
Ñ	UCNT	Uppercase N with tilde	0101 0001	081	51	121
Ò	UCOG	Uppercase O with grave accent	0101 0010	082	52	122
Ó	UCOA	Uppercase O with acute accent	0101 0011	083	53	123

Table B-1  
Prime Extended Character Set – Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
Ô	UCOC	Uppercase O with circumflex	0101 0100	084	54	124
Õ	UCOT	Uppercase O with tilde	0101 0101	085	55	125
Ö	UCOD	Uppercase O with diaeresis	0101 0110	086	56	126
×	MULT	Multiplication sign used in mathematics	0101 0111	087	57	127
Ø	UCOO	Uppercase O with oblique line	0101 1000	088	58	130
Ù	UCUG	Uppercase U with grave accent	0101 1001	089	59	131
Ú	UCUA	Uppercase U with acute accent	0101 1010	090	5A	132
Û	UCUC	Uppercase U with circumflex	0101 1011	091	5B	133
Ü	UCUD	Uppercase U with diaeresis	0101 1100	092	5C	134
Ý	UCYA	Uppercase Y with acute accent	0101 1101	093	5D	135
Þ	UTHN	Uppercase Icelandic letter <u>Thorn</u>	0101 1110	094	5E	136
ß	LGSS	Lowercase German letter double <u>s</u>	0101 1111	095	5F	137
à	LCAG	Lowercase a with grave accent	0110 0000	096	60	140
á	LCAA	Lowercase a with acute accent	0110 0001	097	61	141
â	LCAC	Lowercase a with circumflex	0110 0010	098	62	142
ã	LCAT	Lowercase a with tilde	0110 0011	099	63	143
ä	LCAD	Lowercase a with diaeresis	0110 0100	100	64	144
å	LCAR	Lowercase a with ring above	0110 0101	101	65	145
æ	LCAE	Lowercase diphthong <u>ae</u>	0110 0110	102	66	146
ç	LCCC	Lowercase c with cedilla	0110 0111	103	67	147
è	LCEG	Lowercase e with grave accent	0110 1000	104	68	150
é	LCEA	Lowercase e with acute accent	0110 1001	105	69	151
ê	LCEC	Lowercase e with circumflex	0110 1010	106	6A	152

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
ë	LCED	Lowercase e with diaeresis	0110 1011	107	6B	153
ì	LCIG	Lowercase i with grave accent	0110 1100	108	6C	154
í	LCIA	Lowercase i with acute accent	0110 1101	109	6D	155
î	LCIC	Lowercase i with circumflex	0110 1110	110	6E	156
ï	LCID	Lowercase i with diaeresis	0110 1111	111	6F	157
ð	LETH	Lowercase Icelandic letter <u>Eth</u>	0111 0000	112	70	160
ñ	LCNT	Lowercase n with tilde	0111 0001	113	71	161
ò	LCOG	Lowercase o with grave accent	0111 0010	114	72	162
ó	LCOA	Lowercase o with acute accent	0111 0011	115	73	163
ô	LCOC	Lowercase o with circumflex	0111 0100	116	74	164
õ	LCOT	Lowercase o with tilde	0111 0101	117	75	165
ö	LCOD	Lowercase o with diaeresis	0111 0110	118	76	166
÷	DIV	Division sign used in mathematics	0111 0111	119	77	167
ø	LCOO	Lowercase o with oblique line	0111 1000	120	78	170
ù	LCUG	Lowercase u with grave accent	0111 1001	121	79	171
ú	LCUA	Lowercase u with acute accent	0111 1010	122	7A	172
û	LCUC	Lowercase u with circumflex	0111 1011	123	7B	173
ü	LCUD	Lowercase u with diaeresis	0111 1100	124	7C	174
ý	LCYA	Lowercase y with acute accent	0111 1101	125	7D	175
þ	LTHN	Lowercase Icelandic letter <u>Thorn</u>	0111 1110	126	7E	176
ÿ	LCYD	Lowercase y with diaeresis	0111 1111	127	7F	177

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	NUL	Null	1000 0000	128	80	200
^A	SOH/TC1	Start of heading	1000 0001	129	81	201
^B	STX/TC2	Start of text	1000 0010	130	82	202
^C	ETX/TC3	End of text	1000 0011	131	83	203
^D	EOT/TC4	End of transmission	1000 0100	132	84	204
^E	ENQ/TC5	Enquiry	1000 0101	133	85	205
^F	ACK/TC6	Acknowledge	1000 0110	134	86	206
^G	BEL	Bell	1000 0111	135	87	207
^H	BS/FE0	Backspace	1000 1000	136	88	210
^I	HT/FE1	Horizontal tab	1000 1001	137	89	211
^J	LF/NL/FE2	Line feed	1000 1010	138	8A	212
^K	VT/FE3	Vertical tab	1000 1011	139	8B	213
^L	FF/FE4	Form feed	1000 1100	140	8C	214
^M	CR/FE5	Carriage return	1000 1101	141	8D	215
^N	SO/LS1	Shift out	1000 1110	142	8E	216
^O	SI/LS0	Shift in	1000 1111	143	8F	217
^P	DLE/TC7	Data link escape	1001 0000	144	90	220
^Q	DC1/XON	Device control 1	1001 0001	145	91	221
^R	DC2	Device control 2	1001 0010	146	92	222
^S	DC3/XOFF	Device control 3	1001 0011	147	93	223
^T	DC4	Device control 4	1001 0100	148	94	224
^U	NAK/TC8	Negative acknowledge	1001 0101	149	95	225
^V	SYN/TC9	Synchronous idle	1001 0110	150	96	226
^W	ETB/TC10	End of transmission block	1001 0111	151	97	227
^X	CAN	Cancel	1001 1000	152	98	230
^Y	EM	End of medium	1001 1001	153	99	231
^Z	SUB	Substitute	1001 1010	154	9A	232
^[	ESC	Escape	1001 1011	155	9B	233
^\ ^]	FS/IS4 GS/IS3	File separator Group separator	1001 1100 1001 1101	156 157	9C 9D	234 235
^^	RS/IS2	Record separator	1001 1110	158	9E	236
^_	US/IS1	Unit separator	1001 1111	159	9F	237
	SP	Space	1010 0000	160	A0	240
!		Exclamation mark	1010 0001	161	A1	241
''		Quotation mark	1010 0010	162	A2	242
#	NUMB	Number sign	1010 0011	163	A3	243
\$	DOLR	Dollar sign	1010 0100	164	A4	244
%		Percent sign	1010 0101	165	A5	245
&		Ampersand	1010 0110	166	A6	246

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
'		Apostrophe	1010 0111	167	A7	247
(		Left parenthesis	1010 1000	168	A8	250
)		Right parenthesis	1010 1001	169	A9	251
*		Asterisk	1010 1010	170	AA	252
+		Plus sign	1010 1011	171	AB	253
,		Comma	1010 1100	172	AC	254
-		Minus sign	1010 1101	173	AD	255
.		Period	1010 1110	174	AE	256
/		Slash	1010 1111	175	AF	257
0		Zero	1011 0000	176	B0	260
1		One	1011 0001	177	B1	261
2		Two	1011 0010	178	B2	262
3		Three	1011 0011	179	B3	263
4		Four	1011 0100	180	B4	264
5		Five	1011 0101	181	B5	265
6		Six	1011 0110	182	B6	266
7		Seven	1011 0111	183	B7	267
8		Eight	1011 1000	184	B8	270
9		Nine	1011 1001	185	B9	271
:		Colon	1011 1010	186	BA	272
;		Semicolon	1011 1011	187	BB	273
<		Less than sign	1011 1100	188	BC	274
=		Equal sign	1011 1101	189	BD	275
>		Greater than sign	1011 1110	190	BE	276
?		Question mark	1011 1111	191	BF	277
@	AT	Commercial at sign	1100 0000	192	C0	300
A		Uppercase A	1100 0001	193	C1	301
B		Uppercase B	1100 0010	194	C2	302
C		Uppercase C	1100 0011	195	C3	303
D		Uppercase D	1100 0100	196	C4	304
E		Uppercase E	1100 0101	197	C5	305
F		Uppercase F	1100 0110	198	C6	306
G		Uppercase G	1100 0111	199	C7	307
H		Uppercase H	1100 1000	200	C8	310
I		Uppercase I	1100 1001	201	C9	311
J		Uppercase J	1100 1010	202	CA	312
K		Uppercase K	1100 1011	203	CB	313
L		Uppercase L	1100 1100	204	CC	314
M		Uppercase M	1100 1101	205	CD	315
N		Uppercase N	1100 1110	206	CE	316

**Table B-1**  
**Prime Extended Character Set – Continued**

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
O		Uppercase O	1100 1111	207	CF	317
P		Uppercase P	1101 0000	208	D0	320
Q		Uppercase Q	1101 0001	209	D1	321
R		Uppercase R	1101 0010	210	D2	322
S		Uppercase S	1101 0011	211	D3	323
T		Uppercase T	1101 0100	212	D4	324
U		Uppercase U	1101 0101	213	D5	325
V		Uppercase V	1101 0110	214	D6	326
W		Uppercase W	1101 0111	215	D7	327
X		Uppercase X	1101 1000	216	D8	330
Y		Uppercase Y	1101 1001	217	D9	331
Z		Uppercase Z	1101 1010	218	DA	332
[	LBKT	Left bracket	1101 1011	219	DB	333
\	REVS	Reverse slash, backslash	1101 1100	220	DC	334
]	RBKT	Right bracket	1101 1101	221	DD	335
^	CFLX	Circumflex	1101 1110	222	DE	336
_		Underline, underscore	1101 1111	223	DF	337
`	GRAV	Left single quote, grave accent	1110 0000	224	E0	340
a		Lowercase a	1110 0001	225	E1	341
b		Lowercase b	1110 0010	226	E2	342
c		Lowercase c	1110 0011	227	E3	343
d		Lowercase d	1110 0100	228	E4	344
e		Lowercase e	1110 0101	229	E5	345
f		Lowercase f	1110 0110	230	E6	346
g		Lowercase g	1110 0111	231	E7	347
h		Lowercase h	1110 1000	232	E8	350
i		Lowercase i	1110 1001	233	E9	351
j		Lowercase j	1110 1010	234	EA	352
k		Lowercase k	1110 1011	235	EB	353
l		Lowercase l	1110 1100	236	EC	354
m		Lowercase m	1110 1101	237	ED	355
n		Lowercase n	1110 1110	238	EE	356
o		Lowercase o	1110 1111	239	EF	357
p		Lowercase p	1111 0000	240	F0	360
q		Lowercase q	1111 0001	241	F1	361
r		Lowercase r	1111 0010	242	F2	362
s		Lowercase s	1111 0011	243	F3	363
t		Lowercase t	1111 0100	244	F4	364

Table B-1  
Prime Extended Character Set – Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
u		Lowercase u	1111 0101	245	F5	365
v		Lowercase v	1111 0110	246	F6	366
w		Lowercase w	1111 0111	247	F7	367
x		Lowercase x	1111 1000	248	F8	370
y		Lowercase y	1111 1001	249	F9	371
z		Lowercase z	1111 1010	250	FA	372
{	LBCE	Left brace	1111 1011	251	FB	373
	VERT	Vertical line	1111 1100	252	FC	374
}	RBCE	Right brace	1111 1101	253	FD	375
~	TIL	Tilde	1111 1110	254	FE	376
	DEL	Delete	1111 1111	255	FF	377

---

# Index

# Index

## Symbols

!, 2-19, 3-26  
!!, 2-19, 3-26  
#, 2-23, 3-1  
#off, 2-23, 3-1  
#on, 2-23, 3-2  
? help option, 4-1, 4-5

## Numbers

2d, 2-23, 3-2  
2doff, 2-23, 3-2  
2don, 2-23, 3-2

## Keybindings

**Backspace** **Delete**, 3-31  
**Ctrl @**, 2-9, 3-20  
**Ctrl \_**, 2-2, 3-16, 4-1, 6-6  
**Ctrl ?**, 2-2, 4-1  
**Ctrl A**, 2-2, 3-4, 4-1  
**Ctrl C**, 2-2, 3-11, 4-1  
**Ctrl D**, 2-2, 3-9, 4-1  
**Ctrl L**, 2-2, 4-1  
**Ctrl A**, 2-3, 3-6  
**Ctrl B**, 2-3, 3-4  
**Ctrl C**, 2-14, 3-15, 3-29  
**Ctrl D**, 2-7, 2-20, 3-2, 3-9, 3-21  
**Ctrl E**, 2-3, 3-10  
**Ctrl F**, 2-3, 3-14  
**Ctrl G**, 2-14, 2-26, 3-3 to 3-4, 3-27, 3-39  
**Ctrl H**, 2-7, 2-20, 3-2, 3-31  
**Ctrl I**, 2-12, 3-41, 5-1  
**Ctrl K**, 2-8, 2-20, 3-2, 3-18, 3-21  
**Ctrl L**, 2-4  
    with numeric arguments, 3-29  
**Ctrl M**,  
    use of, 3-34  
**Ctrl N**, 2-3, 2-23, 3-23  
**Ctrl O**, 2-6, 2-12, 3-23  
**Ctrl P**, 1-13, 2-14, 3-6  
**Ctrl Q**, 2-7, 3-2, 5-2, 5-14  
**Ctrl R**, 2-10  
**Ctrl R** or **Esc** **R**, 3-30

**Ctrl S**, 2-10, 3-3, 3-15  
**Ctrl T**, 2-11, 3-41  
**Ctrl U**, 1-5, 2-26, 3-22  
**Ctrl U** **Ctrl @**, 2-9, 2-10, 3-20  
**Ctrl U** **Ctrl X** **Ctrl V**, 2-21, 3-43  
**Ctrl V**, 2-4, 3-23  
**Ctrl W**, 2-8, 2-9, 3-18  
**Ctrl W** on PT45, 3-15  
**Ctrl X** **(**, 2-18, 3-8  
**Ctrl X** **)**, 2-18, 3-14  
**Ctrl X** **[**, 2-5, 3-17  
**Ctrl X** **]**, 2-5, 3-17  
**Ctrl X** **[**, 2-4, 3-6  
**Ctrl X** **]**, 2-4, 3-14  
**Ctrl X** **.**, 2-6, 2-13, 3-39  
**Ctrl X** **=**, 2-24, 3-39  
**Ctrl X** **-**, 2-26, 3-38  
**Ctrl X** **<**, 2-26, 3-5  
**Ctrl X** **>**, 2-26, 3-14  
**Ctrl X** **+**, 2-26, 3-37  
**Ctrl X** **1**, 2-17, 3-21  
**Ctrl X** **2**, 2-17, 3-21  
**Ctrl X** **3**, 2-17, 3-22  
**Ctrl X** **4**, 2-17, 3-32  
**Ctrl X** **A**, 2-9, 3-4  
**Ctrl X** **B**, 2-16, 3-13, 3-21  
**Ctrl X** **B** **Return**, 2-16, 3-21  
**Ctrl X** **Ctrl B**, 2-16, 3-19  
**Ctrl X** **Ctrl C**, 1-13, 2-14, 3-28  
**Ctrl X** **Ctrl E**, 2-19, 3-26  
**Ctrl X** **Ctrl F**, 2-15, 3-13  
**Ctrl X** **Ctrl G**, 2-14, 3-17  
**Ctrl X** **Ctrl H**, 2-8, 3-6  
**Ctrl X** **Ctrl I**, 2-12, 3-18  
**Ctrl X** **Ctrl K**, 2-8, 3-5  
**Ctrl X** **Ctrl L**, 2-11, 3-19, 3-43  
**Ctrl X** **Ctrl O**, 2-7, 3-8  
**Ctrl X** **Ctrl R**, 2-15, 3-13  
**Ctrl X** **Ctrl S**, 1-12, 2-14, 2-15, 3-13, 3-31  
**Ctrl X** **Ctrl T**, 2-25, 3-40  
**Ctrl X** **Ctrl U**, 2-11, 3-41, 3-43  
**Ctrl X** **Ctrl V**, 2-21, 3-42  
**Ctrl X** **Ctrl W**, 1-12, 2-14, 2-15, 3-22, 3-31

**Ctrl X** **Ctrl X**, 2-9, 3-9 to 3-10, 3-18  
**Ctrl X** **Ctrl Z** **<**, 2-9, 3-20  
**Ctrl X** **Ctrl Z** **>**, 2-9, 3-20  
**Ctrl X** **Ctrl Z** **Ctrl A**, 2-4, 3-5  
**Ctrl X** **Ctrl Z** **Ctrl E**, 2-4, 3-14  
**Ctrl X** **Ctrl Z** **Ctrl F**, 2-24, 3-15  
**Ctrl X** **Ctrl Z** **Ctrl G**, 2-14, 3-17  
**Ctrl X** **Ctrl Z** **Ctrl H**, 2-8, 3-5  
**Ctrl X** **Ctrl Z** **Ctrl K**, 2-8, 3-14  
**Ctrl X** **Ctrl Z** **Ctrl V**, 2-25, 3-43  
**Ctrl X** **Ctrl Z** **Ctrl Y**, 2-8, 3-42, 3-44  
**Ctrl X** **Ctrl Z** **A**, 2-9, 3-4  
**Ctrl X** **Ctrl Z** **F**, 2-13, 3-39  
**Ctrl X** **Ctrl Z** **I**, 2-16, 3-17  
**Ctrl X** **Ctrl Z** **K**, 2-8, 3-42  
**Ctrl X** **Ctrl Z** **P**, 2-9, 3-25  
**Ctrl X** **Ctrl Z** **S**, 2-12, 3-7  
**Ctrl X** **D**, 2-22, 3-11  
**Ctrl X** **E**, 2-18, 3-11  
**Ctrl X** **F**, 2-6, 2-13, 3-34  
**Ctrl X** **H**, 2-9, 3-20  
**Ctrl X** **O**, 2-17, 3-24  
**Ctrl X** **P**, 2-9, 3-25  
**Ctrl X** **Q**, 2-7, 3-3, 3-29  
**Ctrl X** **R**, 2-3  
    with numeric arguments, 3-30  
**Ctrl X** **Return**, 2-12, 3-8  
**Ctrl X** **S**, 2-14 to 2-15, 3-31  
**Ctrl X** **U**, 2-22, 3-12  
**Ctrl X** **V**, 2-17, 3-32  
**Ctrl Y**, 2-8, 3-18, 3-44  
**Ctrl Z**, 2-3, 2-23, 3-25  
**Esc** **%**, 2-11, 3-28  
**Esc** **?**, 2-3, 3-11, 4-3  
**Esc** **<**, 2-3, 3-22  
**Esc** **>**, 2-3, 3-22  
**Esc** **^**, 2-7, 3-21  
**Esc** **\**, 2-7, 3-44  
**Esc** **~**, 2-15, 3-41  
**Esc** **@**, 2-9, 3-20  
**Esc** **A**, 2-4, 3-6  
**Esc** **B**, 2-4, 3-5

[Esc] [C], 2-11, 3-7  
 [Esc] [Ctrl D], 2-8, 3-18  
 [Esc] [Ctrl H], 2-8, 3-31  
 [Esc] [Ctrl G], 3-17  
 [Esc] [Ctrl I], 2-12, 3-17  
 [Esc] [Ctrl O], 2-12, 3-38  
 [Esc] [Ctrl V], 2-17, 3-32  
 [Esc] [Ctrl Y], 2-27, 3-44  
 [Esc] [D], 2-8, 3-9  
 [Esc] [E], 2-4, 3-15, 3-44  
 [Esc] [Esc], 2-18, 3-24  
 [Esc] [F], 2-4, 3-15  
 [Esc] [G], 2-4, 2-23, 3-16  
 [Esc] [H], 2-9, 3-20  
 [Esc] [I], 2-12, 3-17  
 [Esc] [K], 2-8, 3-14  
 [Esc] [L], 2-11, 3-20  
 [Esc] [M], 2-4, 3-5  
 [Esc] [N], 2-16, 3-23  
 [Esc] [P], 2-16, 2-22, 3-25  
 [Esc] [Q], 2-5, 2-6, 2-13, 3-13, 3-43  
 [Esc] [R], 2-10, 3-31  
 [Esc] [S], 2-10, 3-3, 3-14  
 [Esc] [SPACE], 2-7, 3-19  
 [Esc] [T], 2-11, 3-40  
 [Esc] [U], 2-11, 3-41  
 [Esc] [V], 2-4, 3-5  
     on PT200 terminal, 1-3  
 [Esc] [W], 2-8, 2-9, 3-8  
 [Esc] [X] Commands, 1-4, 2-26, 3-12  
     with argument of 0, 1-5  
 [Esc] [Y], 2-8, 3-44  
 [Return], 2-12, 2-21, 2-27, 3-8, 3-11  
 [SPACE], 2-21

**A**

## Abbreviations

    on EMACS command line, 1-12  
     PRIMOS in EMACS, 3-26  
     speed-type, 5-1  
     template, 5-3  
 abort\_command, 2-14, 3-3, 3-39  
 abort\_minibuffer, 2-26, 3-4  
 all\_modes\_off, 2-20, 3-4  
 append\_to\_buf, 2-9, 3-4  
 append\_to\_file, 2-9, 3-4

apropos, 2-2 to 2-3, 3-4, 4-1 to 4-2  
 Arguments, numeric, 1-5, 3-16, 3-22 to 3-23, 3-25

**B**

back\_char, 2-3, 3-4  
 back\_page, 2-4, 3-5  
 back\_place\_holder, 2-26, 3-5, 5-13 to 5-14  
 back\_tab, 2-12, 3-5  
 back\_to\_nonwhite, 2-4, 3-5  
 backward\_clause, 2-4, 3-5  
 backward\_kill\_clause, 2-8, 3-5  
 backward\_kill\_line, 2-8, 3-5  
 backward\_kill\_sentence, 2-8, 3-6  
 backward\_para, 2-4, 3-6  
 backward\_sentence, 2-4, 3-6  
 back\_word, 2-4, 3-5  
 begin\_line, 2-3, 3-6  
 Binding commands and functions, 1-4, 3-33, 3-34, 6-2, 6-6 to 6-7, 8-23  
 Break character  
     *see*: PRIMOS break  
 Buffer and window commands  
     buffers, 2-16  
     windows, 2-16  
 buffer\_info function, 6-8  
 Buffers, 1-1, 1-8 to 1-9, 2-15 to 2-16, 3-4, 3-19, 3-21, 3-23, 3-25 to 3-26, 3-29, 3-31  
     setting tabs, 3-35  
     size of, 3-17, 3-24

**C**

C mode, 8-7  
     abbreviations and templates, 8-10  
     additional information, 8-13  
     commands and keybindings, 8-9  
     turning off, 8-8  
     turning on, 8-7, 8-9  
 Capabilities, TERMCAP  
     *see*: TERMCAP capabilities  
 capinitial, 2-11, 3-7  
 Carriage return character, 2-12, 2-20, 3-2, 3-8, 3-44  
 case?, 2-11, 3-7  
 Case conversion, 2-11, 3-19  
 Case matching, 3-7  
 case\_off, 2-10, 3-7  
 case\_on, 2-10, 3-7  
 case\_replace?, 2-11, 3-7  
 case\_replace\_off, 2-11, 3-7  
 case\_replace\_on, 2-11, 3-7  
 CBL compiler, 8-1  
     options, 8-31

center\_line, 2-12, 3-7  
 Character sequences, 1-4  
 clerical\$ user type, 6-3  
 COBOL mode, 5-14, 6-3, 8-1  
     additional information, 8-2  
     commands and keybindings, 8-2  
     commands disabled in, 8-6  
     entering text, 8-2  
     fundamental mode commands in, 8-2  
     keybindings, 8-2  
     turning off, 8-2  
     turning on, 8-1  
 collect\_macro, 2-18, 3-8, 6-8  
 .COMI file, execution of, 3-27  
 Commands for slow terminals, 2-25  
 Common LISP mode  
     additional information, 8-26  
     commands and keybindings, 8-26  
     turning off, 8-26  
     turning on, 8-26  
 Compile command  
     compiler options, 8-31  
     compiling programs from EMACS, 8-29  
     debugging programs, 8-29, 8-32  
     internal variables, 8-30  
     initializing variables, 8-31  
 Continuous lines environment (two-dimensional mode)  
     *see*: Environments  
 Control key, 1-3  
 copy\_region, 2-8 to 2-9, 3-8  
 cr, 2-12, 2-20, 3-8  
 cret\_indent\_relative, 2-12, 3-8  
 Cursor movement commands, 2-3  
 Cursor-function/number modes  
     *see*: Modes

**D**

date, 2-24, 3-8  
 Debugging programs, 8-29, 8-32  
 default\_tabs, 2-13, 3-8  
 Defun statement, 6-5  
 delete\_blank\_lines, 2-7, 3-8  
 delete\_buffer, 2-7, 3-9  
 delete\_char, 1-4, 2-7, 2-20, 3-9, 6-1, 6-6  
 delete\_region, 2-7, 3-9  
 delete\_word, 2-8, 3-9  
 Deleting and restoring text, 2-7, 3-4, 3-18, 3-44  
 Delimiters, sentence, 3-6, 3-14 to 3-15  
 Delimiters, word, 3-15  
 describe, 2-2 to 2-3, 3-9, 4-1, 4-3 to 4-4  
 Dispatch table  
     *see*: Modes

display\_buffer, 2-24, 3-9  
 display\_debug, 2-24, 3-9  
 display\_terminal, 2-24, 3-9  
 display\_window, 2-24, 3-10  
 dt, 2-24, 3-10  
 dump\_file, 2-25, 3-10, 6-2  
 Dynamic segments, 2-15

## E

Editing commands, 2-5  
 abort, break, and reexecute commands, 2-14  
 case conversion, 2-11  
 deleting and restoring text, 2-7  
 formatting, 2-13  
 inserting new lines, 2-12  
 inserting text, 2-6  
 nonalphabetic characters, 2-7, 3-1  
 ring of marks, 2-10  
 saving text and exiting from EMACS, 2-14  
 searching and replacing, 2-10, 3-3  
 tabs and indentation, 2-12  
 the mark and the region, 2-9  
 transposition, 2-11, 3-41  
 .EFASL, 6-2  
 EMACS command line, 1-9  
 abbreviations, 1-12  
 options, 1-10, 5-6  
 EMACS commands, 2-1  
 aborting, 3-3  
 conventions, 1-3  
 cursor movement, 2-3  
 definition of, 1-1  
 editing, 2-5  
 extended, 1-4  
 file management, 2-15  
 macros, 2-17  
 modes, general, 2-19  
 numeric arguments, 1-5  
 online help, 2-2  
 PRIMOS commands, 2-19  
 screen display, 2-4  
 ten basic commands, 1-14  
 uppercase or lowercase letters, 3-1  
 EMACS language modes, 8-1  
 C mode, 8-7  
 CBL compiler, 8-1  
 C compiler, 8-9  
 COBOL mode, 8-1  
 Common LISP mode, 8-26  
 compiling COBOL programs, 8-2  
 compiling FORTRAN programs, 8-15  
 compiling VRPG programs, 8-19  
 FORTRAN mode, 8-13

LISP mode, 8-24  
 Listener mode, 8-28  
 RPG mode, 8-16  
 EMACS screen, 1-6  
 displaying extended lines, 2-5, 3-16  
 EMACS status line, 1-6, 1-8, 2-20, 3-16, 3-33, 5-8  
 freezing display, 3-40  
 line numbers on EMACS screen, 1-7  
 messages on EMACS screen, 1-7  
 minibuffer, 1-6, 1-9  
 moving display, 3-5, 3-23  
 organization of, 1-6  
 shifting display left, 3-16 to 3-17, 3-33  
 shifting display right, 2-5, 3-17  
 splitting screen vertically, 3-43  
 text area, 1-6  
 EMACS Standard User Interface (SUI), 1-1, 1-3, 1-11  
 EMACS Standard User Interface with Extensions (SUIX), 1-1, 1-3, 1-11  
 EMACS Status Line  
   *see*: EMACS Screen  
 EMACS terms, common, 1-1  
 .EM.CPL files, 1-14  
 .EM.n files, 1-14  
 end\_line, 2-3, 3-10  
 Entering EMACS, 1-9  
 Entry, TERMCAP  
   *see*: TERMCAP entry  
 Environments, 2-23  
   continuous lines environment (two-dimensional mode), 2-23, 3-2  
   line numbering, 2-23, 3-1  
 Escape key, 1-3, 1-5  
 europe\_dt, 2-24, 3-10  
 exchange\_mark, 2-9, 3-9 to 3-10, 3-18  
 execute\_macro, 2-18, 3-11  
 Exiting from EMACS, 1-13, 2-14  
 exit\_minibuffer, 2-27, 3-11  
 expand\_macro, 2-18, 3-11  
 explain\_key, 2-2 to 2-3, 3-11, 4-1 to 4-2  
 explore, 2-22, 3-11  
 Explore mode  
   *see*: Modes  
 Explore mode options, 2-22, 3-12  
 extend\_command, 2-26, 3-12  
 Extended commands, 1-4

## F

Fasdump format, 2-25, 3-10, 6-2  
 Fasload, 3-19, 6-7  
 File hooks, 2-19

File hooks mechanism, 6-3  
 clerical\$ user type, 6-3 to 6-4  
 creating and changing, 6-5  
 no\_file\_hooks\$ user type, 6-3  
 programmer\$ user type, 6-3  
 setq function, 6-4  
 suffix for programmer\$ user type, 6-4  
 File management commands, 2-15  
 Filename, valid PRIMOS, 1-10  
 file\_output buffer, 3-26 to 3-27  
 Fill mode  
   *see*: Modes  
 fill\_off, 2-6, 2-21, 3-12  
 fill\_on, 2-6, 2-21, 3-13  
 fill\_para, 2-6, 2-13, 3-13, 3-39, 3-43  
 find\_file, 2-15, 3-13  
 finish\_macro, 2-18, 3-14, 6-8  
 Formatting commands, 2-13  
 FORTRAN mode, 8-13  
   additional information, 8-15  
   compiling programs, 8-15  
   entering text, 8-14  
   turning off, 8-15  
   turning on, 8-14  
   useful commands, 8-15  
 forward\_char, 2-3, 3-14  
 forward\_clause, 2-4, 3-14  
 forward\_kill\_clause, 2-8, 3-14  
 forward\_kill\_sentence, 2-8, 3-14  
 forward\_para, 2-4, 3-14  
 forward\_place\_holder, 2-26, 3-14, 5-12, 5-14  
 forward\_search\_again, 2-10, 3-15  
 forward\_search\_command, 2-10, 3-3, 3-14  
 forward\_sentence, 2-4, 3-15  
 forward\_word, 2-4, 3-15  
 Function key keypaths, lists of, 6-7  
 Function keys, 1-1, 1-3, 3-15  
 Functions, 1-2 to 1-3  
   binding to Esc sequence, 3-34  
   binding to terminal key, 6-6  
   buffer\_info, 6-8  
   collect\_macro, 6-8  
   finish\_macro, 6-8  
   found\_file\_hook, 6-3  
   help\_on\_tap, 6-8  
   next\_line\_command, 1-4  
   overlay, 2-20, 3-24  
   self\_insert, 2-6, 3-33  
   setq, 6-2  
   toggle\_overlay\_off, 6-8  
   toggle\_overlay\_on, 6-8  
 fundamental command, 2-20, 3-15  
 Fundamental mode  
   *see*: Modes

**G**

get\_bufname, 2-16, 2-24, 3-15  
 get\_filename, 2-24, 3-15  
 get\_tab, 2-13, 3-15  
 global\_tabs, 2-13, 3-16  
 goto\_line, 2-4, 2-23, 3-16

**H**

hcol, 2-5, 3-16, 3-33  
 Help commands, 2-2, 4-1  
   describe command options, 4-4  
   description of commands and functions, 4-3  
   EMACS help options, 4-5  
   list of last 20 characters you typed, 4-4  
   what a command does, 4-2  
   which command to use, 4-2  
 help\_on\_tap, 2-2, 3-16, 4-1, 6-8  
 horiz\_left, 2-5, 3-17, 3-43  
 Horizontal movement of screen, 2-5, 3-16  
 horiz\_right, 2-5, 3-17, 3-43  
 hscroll, 2-5, 3-17

**I**

ignore\_prefix, 2-14, 3-17  
 indent\_relative, 2-12, 3-17  
 indent\_to\_fill\_prefix, 2-12, 3-17  
 Information commands, 2-24  
 insert\_buf, 2-16, 3-17  
 insert\_file, 2-15, 3-18  
 Inserting new lines, 2-12  
 Inserting text, 2-6  
 insert\_tab, 2-12, 3-18  
 insert\_version, 2-24, 3-18

**K**

Keybindings, 1-2, 1-4  
   creating, 3-33  
   customized, 6-6  
   fundamental mode, 3-15  
   in COBOL mode, 8-2  
   in LISP mode, 8-24  
   lists of function key keypaths, 6-7  
   toggles for, 6-7  
   uppercase or lowercase letters, 3-1  
 Keypath, 1-2, 1-4  
   specifying, 3-34  
 Kill ring, 2-7 to 2-8, 3-18, 3-42, 3-44  
   recalling text, 3-44  
 kill\_line, 2-8, 2-20, 3-18  
 kill\_region, 2-8 to 2-9, 3-18  
 kill\_rest\_of\_buffer, 2-8, 3-18

**L**

L help option, 4-1, 4-4  
 Language modes  
   *see*: EMACS language modes  
 leave\_one\_white, 2-7, 3-19  
 Library commands, 2-25  
 Library files  
   creating user, 6-2  
   customized, 6-1  
   .EFASL suffix, 6-2  
   .EM suffix, 6-2  
   example of user, 6-1  
   fasdump operation, 6-2  
   file hook categories, 6-3  
   menus, 6-8  
   PEEL statements in, 6-1  
   saving and executing user, 6-2  
   -ULIB option, 6-2  
 Line numbering, 2-23  
   *see also*: Environments  
 Line numbers on EMACS screen  
   *see*: EMACS Screen  
 LISP mode  
   commands and keybindings, 8-24  
   Common LISP mode, 8-26  
   Listener mode, 8-28  
   turning off, 8-24  
   turning on, 8-24  
 list\_buffers, 2-16, 3-19  
 Listener mode  
   additional information, 8-28  
   commands and keybindings, 8-28  
   turning off, 8-28  
   turning on, 8-28  
 load\_compiled, 2-25, 3-19, 6-3  
 load\_pl\_source, 2-25, 3-19, 6-2  
 local\_tabs, 2-13, 3-19  
 Locked keyboard condition, 1-14  
 Logout, forced, 1-13  
 lowercase\_region, 2-11, 3-19, 3-43  
 lowercase\_word, 2-11, 3-20

**M**

Macros, 1-2, 2-17, 3-8, 3-11  
   keybindings, 3-15  
   with Global Replace key, 3-30  
 Margins, 2-13  
 mark, 2-9 to 2-10, 3-10, 3-18, 3-20, 3-25, 3-35, 3-44  
 mark\_bottom, 2-9, 3-20  
 mark\_end\_of\_word, 2-9, 3-20  
 mark\_para, 2-9, 3-20  
 mark\_top, 2-9, 3-20  
 mark\_whole, 2-9, 3-20

Menus in library files, 6-8  
 merge\_lines, 2-7, 3-21  
 Messages on EMACS screen  
   *see*: EMACS Screen  
 Minibuffer, 1-2 to 1-3, 1-9  
   current buffer information, 3-39  
   margin display, 3-39  
 Miscellaneous commands, 2-26  
 Mod prefix, 1-12  
 Modes, 1-2  
   binding a function to a keypath, 3-34  
   cursor-function/number, 2-22  
   dispatch table, 2-20  
   explore mode, 2-22, 3-11 to 3-12  
   fill mode, 2-6, 2-13, 2-19, 2-21, 3-13, 3-34, 3-39, 6-4  
   fundamental mode, 1-3, 2-19, 3-15, 8-1  
   fundamental mode on PT200, 2-23  
   general, 2-19  
   language modes  
     *see*: EMACS language modes  
   number, 2-22  
   overlay mode, 2-20, 3-2, 3-24, 6-3, 6-7 to 6-8  
   setting with toggle, 6-8  
   view mode, 2-21, 3-41 to 3-43  
 mod\_one\_window, 2-17, 3-21  
 mod\_select\_buf, 2-16, 3-21, 3-33  
 mod\_split\_window, 2-17, 3-21, 3-38  
 mod\_split\_window\_stay, 2-17, 3-22, 3-39  
 mod\_write\_file, 1-12, 2-14 to 2-15, 3-22  
 move\_bottom, 2-3, 3-22  
 move\_top, 2-3, 3-22  
 multiplier, 2-26, 3-22

**N**

New lines, inserting, 2-12, 3-4  
 new\_features, 2-24, 3-22  
 next\_buf, 1-3, 2-16, 3-23  
 next\_line\_command, 1-4, 2-3, 2-23, 3-23, 6-6  
 next\_page, 2-4, 3-23  
 Nonalphabetic characters, 2-7, 3-1  
 Nonprinting characters, 3-2, 3-44  
 -NOXOFF, 1-11, 3-3, 7-3  
 Num lock key, 2-22

**O**

one\_window, 2-17, 3-23  
 open\_line, 2-6, 2-12, 3-23  
 Options on EMACS command line, 1-9 to 1-10  
   -ECHO\_CPL, 1-10  
   -HEIGHT, 1-10

Options on EMACS command line  
(continued)

- HELP, 1-10
- NOXOFF, 1-11, 3-3
- NULIB, 1-11
- SAVE\_SCREEN, 1-11
- SPDT, 1-11, 5-6, 8-13
- SPEED, 1-11
- SUI, 1-11
- SUIX, 1-11
- TTP, 1-10, 1-12, 7-2 to 7-3
- ULIB, 1-11, 6-3
- WIDTH, 1-11
- XOFF, 1-11

other\_window, 2-17, 3-24

## Overlay mode

*see:* Modes

overlayer, 2-20, 3-24  
 overlay\_off, 2-6, 2-20, 3-24  
 overlay\_on, 2-6, 2-20, 3-24  
 overlay\_rubout, 2-20, 3-24

## P

Passwords, 3-13, 3-29  
 Pathname, 1-8 to 1-9, 5-6  
 PEEL statements, 1-3, 2-2, 2-18, 2-25,  
   3-10 to 3-11, 3-19, 3-24, 6-1,  
   8-19, 8-29  
   defun statement, 6-5  
   for Common Lisp mode, 8-24  
   function for programmer\$ user type,  
   6-5  
   select statement, 6-5  
 PF12 key, 6-1, 6-7  
 pl, 2-18, 3-24, 3-33, 8-29  
 PL: prompt, 1-3, 3-24  
 Placeholders, Speed-type, 5-3, 5-9  
 pl\_minibuffer, 2-18, 3-24  
 Point, 1-4, 2-3, 2-9, 3-9 to 3-10  
 popmark, 2-10, 3-25  
 Prefix, 1-2  
 prepend\_to\_buf, 2-9, 3-25  
 prepend\_to\_file, 2-9, 3-25  
 Pretty printing, 8-7, 8-27  
 prev\_buf, 2-16, 2-22, 3-25  
 prev\_line\_command, 2-3, 2-23, 3-25  
 Prime EMACS Extension Language  
   statements  
   *see:* PEEL statements  
 PRIMOS abbreviations, 3-26  
 PRIMOS break, 2-14, 3-6  
 PRIMOS command execution, 2-19, 3-26  
 PRIMOS "start print" command, 1-11, 2-7,  
   3-3  
 PRIMOS "stop print" command, 1-11, 3-3

primos\_command, 2-19, 3-26  
   options to, 3-26  
 primos\_external, 2-19, 3-26 to 3-27  
 primos\_internal\_como, 2-19, 3-26 to 3-27  
 primos\_internal\_screen, 2-19, 3-26 to 3-27  
 programmer\$ user type, 6-5  
 PST 100 terminal, 1-10, 7-3  
   TERMCAP entry, 7-12  
 PT200 terminal, 1-10, 7-3  
 PT200W terminal, 1-10  
 PT45 SEND key, 3-15  
 PT45 terminal, 1-10, 7-3  
 pushmark, 2-10, 3-28

## Q

^q\_quote\_command, 2-7, 3-2, 3-29, 5-2,  
   5-14  
 query\_replace, 2-11, 3-28  
 quit, 1-13, 2-14, 3-28  
 Quitting EMACS, 1-13  
 quote\_command, 2-7, 3-29

## R

read\_file, 2-15, 3-13, 3-29  
 Recovering from an error, 1-13  
 REENTER, 1-13  
 eexecute, 2-14, 3-29  
 refresh, 2-4, 3-29  
 Regions, 2-9, 3-4, 3-9, 3-20, 3-25, 3-28  
 reject, 3-30  
 repaint, 2-3, 3-30  
 replace, 2-11, 3-30  
 Replacing character strings, 3-28  
 reset, 2-5, 3-30  
 reverse\_search, 4-3  
 reverse\_search\_again, 2-10, 3-31  
 reverse\_search\_command, 2-10, 3-30  
 Ring of marks, 2-10, 3-20  
 RPG mode, 8-22  
   additional information, 8-19  
   compiling VRPG programs, 8-19  
   configuring RPG mode, 8-19  
   entering text, 8-18  
   fundamental mode commands, 8-18  
   help messages, 8-20  
   library files, 8-22  
   specification sheet templates, 8-16,  
   8-22  
   turning off, 8-19  
   turning on, 8-16  
 rubout\_char, 2-7, 2-20, 3-31  
 rubout\_word, 2-8, 3-31

## S

save\_all\_files, 1-13, 2-14 to 2-15, 3-31  
 save\_file, 1-12, 2-14 to 2-15, 3-13, 3-31  
 save\_tab, 2-13, 3-15, 3-32  
 saving text, 1-12, 2-14  
 Screen display commands, 2-4  
   horizontal movement, 2-5  
   vertical movement, 2-4  
   wide-screen, 2-5  
 Screen, EMACS  
   *see:* EMACS Screen  
 scroll\_other\_backward, 2-17, 3-32  
 scroll\_other\_forward, 2-17, 3-32  
 search\_fd, 4-3  
 Searching and replacing commands, 2-10,  
   3-3  
 Segments, dynamic, 2-15  
 Select statement, 6-5  
 select\_any\_window, 2-17, 3-32  
 select\_buf, 2-16, 3-13, 3-21, 3-32  
 self\_insert function, 2-6, 3-24, 3-33  
 Separators, line, 1-5, 3-18, 3-23, 3-25,  
   3-31  
 Separators, word, 3-23, 3-25, 5-1  
   changing speed-type, 5-13  
   in COBOL mode, 8-5  
   speed-type for fundamental mode, 5-8  
 setfit, 2-13, 3-37  
 set\_hscroll, 2-5, 3-33  
 set\_key, 2-18, 3-33, 6-8  
 set\_left\_margin, 2-6, 2-13, 3-17, 3-33  
 setmark, 2-9, 3-35  
 set\_mode, 2-20, 3-33  
 set\_mode\_key, 2-20, 3-34  
 set\_permanent\_key, 2-18, 3-33 to 3-34,  
   6-2, 6-6, 6-8  
 setq function, 6-2, 6-4, 8-31  
 setq\_user\_type\$, 6-4  
 set\_right\_margin, 2-6, 2-13, 3-34  
 set\_tab, 2-12, 3-35  
 settab, 2-12, 3-35  
 set\_tabs, 2-12, 3-35  
 settabs\_from\_table, 2-13, 3-36  
 set\_user\_type, 2-25, 3-37  
 ^s\_forward\_search\_command, 2-10, 3-3  
 Slow terminals, 2-25, 3-40  
 sort\_dt, 2-24, 3-37  
 spd\_add, 2-26, 3-37, 5-2, 5-14  
 spd\_add\_modal, 2-26, 3-37, 5-3, 5-14  
 spd\_add\_region, 2-26, 3-37, 5-3, 5-15  
 spd\_compile, 2-26, 3-37, 5-13, 5-15  
 spd\_delete, 2-26, 3-37, 5-7, 5-15  
 spd\_list, 2-26, 3-37, 5-15  
 spd\_list\_all, 2-26, 3-37, 5-2, 5-4, 5-8, 5-16  
   to 5-17  
 spd\_list\_file, 2-26, 3-38, 5-8, 5-17

spd\_load\_file, 2-26, 3-38, 5-6, 5-13, 5-15, 5-17  
 spd\_off, 2-25, 3-38, 5-7, 5-17  
 spd\_on, 2-25, 3-38, 5-2, 5-7, 5-17  
 spd\_save\_file, 2-26, 3-38, 5-4, 5-17  
 -SPDT, 1-11, 5-5 to 5-6, 8-13  
 spd\_unexpand, 2-26, 3-38, 5-18  
 Special characters, displaying, 2-7, 3-2  
 Speed-type abbreviations, 5-1
 

- adding and deleting interactively, 5-7
- adding templates to source file, 5-9
- changing separators, 5-13
- compiling and loading the source file, 5-13
- creating interactively, 5-2
- creating templates interactively, 5-3
- defining a region as, 5-3
- defining for language modes, 5-13 to 5-14
- editing source files, 5-7
- SPDT option, 1-11, 5-6, 8-13
- .ESPD suffix, 5-4 to 5-6, 5-13, 5-15, 5-17
- expansion, 5-1
- formatting rules for source file, 5-8
- formatting rules for templates, 5-9
- loading from EMACS, 5-6
- loading with -SPDT command line option, 5-6
- placeholder declaration, 5-12
- placeholders, 5-3, 5-9
- primary file, 5-17
- saving during an EMACS session, 5-4
- saving in a file as you exit, 5-4
- separators, 5-1
- .spd\_list buffer, 5-8, 5-16
- template expansion, 5-12
- templates for RUNOFF code, 5-9
- using saved abbreviations in .ESPD file, 5-5

 Speed-type commands, 2-25, 3-37, 5-13  
 split\_line, 2-12, 3-38  
 split\_window, 2-17, 3-38  
 split\_window\_stay, 2-17, 3-38  
 START, 1-13  
 Status Line  
*see: EMACS Screen*

**T**

tablist, 2-12, 3-39  
 Tabs and indentation, 2-12  
 Tabs, setting, 3-32, 3-35, 3-39  
 take\_left\_margin, 2-6, 2-13, 3-39  
 take\_right\_margin, 2-13, 3-39  
 tell\_left\_margin, 2-6, 2-13, 3-39

tell\_modes, 2-20, 2-24, 3-39  
 tell\_position, 2-24, 3-39  
 tell\_right\_margin, 2-6, 2-13, 3-40  
 TERMCAP, 7-1
 

- alaises, 7-12
- defining global variables, 7-2
- disclaimer, 7-1
- Prime TERMCAP database, 7-1, 7-3, 7-11

 TERMCAP capabilities, 7-1, 7-4
 

- alphabetical list of, 7-13
- basic terminal features, 7-4 to 7-6
- Boolean data types, 7-4
- control characters, 7-10
- cursor motion, 7-6
- delay padding, 7-10
- numeric data types, 7-4
- placeholder sequence, 7-10
- screen movement, 7-4 to 7-6
- screen update, 7-4 to 7-5, 7-7
- software control, 7-4 to 7-5, 7-9
- special characteristics, 7-4 to 7-5, 7-9
- string data types, 7-4

 TERMCAP entry, 7-1
 

- adding to TERMCAP database, 7-1, 7-11
- building and testing, 7-11
- Prime PST 100 terminal, 7-12
- structure, 7-12

 .TERMCAP\$ global variable, 7-2, 7-11  
 Terminal status, 1-8  
 Terminal type, 1-9 to 1-10, 7-3
 

- how to indicate, 7-2

 .TERMINALTYPE\$ global variable, 1-10, 7-2, 7-11  
 Tilde character, 3-13, 3-29  
 toggle\_overlay\_off, 6-8  
 toggle\_overlay\_on, 6-8  
 toggle\_redisplay, 2-25, 3-40  
 Toggles, 6-7  
 transpose\_word, 2-11, 3-40  
 Transposition, 2-11, 3-41  
 trim\_date, 2-24, 3-40  
 trim\_dt, 2-24, 3-40  
 -TTP, 1-10, 1-12, 7-2, 7-3  
 twiddle, 2-11, 3-41  
 type\_tab, 2-12, 3-41, 5-1

**U**

unmodify, 2-15, 3-41  
 untidy, 2-13, 3-39, 3-41, 3-43  
 uppercase\_region, 2-11, 3-41, 3-43  
 uppercase\_word, 2-11, 3-41  
 User type, 2-19, 6-5  
 user\_type\$ variable, 6-2, 6-4

**V**

view, 2-21, 3-41  
 View mode, 3-43
 

- see also: Modes*

 View mode options, 2-21, 3-42  
 view\_file, 2-21, 3-42  
 view\_kill\_ring, 2-8, 3-42, 3-44
 

- with numeric arguments, 3-43

 view\_lines, 2-25, 3-40, 3-43  
 view\_off, 2-21, 3-43  
 view\_on, 2-21, 3-43  
 vsplit, 2-17, 3-43

**W**

wallpaper, 2-24, 3-43  
 which\_tabs, 2-13, 3-43  
 white\_delete, 2-7, 3-44  
 Wide-screen display, 2-5  
 Windows, 1-2, 2-16, 3-21, 3-24, 3-32  
 wrap, 2-21, 3-44  
 write\_file, 1-12, 2-14 to 2-15, 3-44

**X**

-XOFF, 1-11

**Y**

yank\_kill\_text, 2-8, 3-42, 3-44  
 yank\_minibuffer, 2-27, 3-44  
 yank\_region, 2-8, 3-44  
 yank\_replace, 2-8, 3-44



# Surveys

---

**READER RESPONSE FORM**

**EMACS Reference Guide**

**DOC5026-2LA**

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate this document for overall usefulness?

*excellent*    *very good*    *good*    *fair*    *poor*

2. What features of this manual did you find most useful?

---

---

---

---

---

3. What faults or errors in this manual gave you problems?

---

---

---

---

---

4. How does this manual compare to equivalent manuals produced by other computer companies?

*Much better*       *Slightly better*       *About the same*  
 *Much worse*       *Slightly worse*       *Can't judge*

5. Which other companies' manuals have you read?

---

---

Name: \_\_\_\_\_

Position: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Postal Code: \_\_\_\_\_



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

**BUSINESS REPLY MAIL**

Postage will be paid by:



Attention: Technical Publications  
Bldg 21  
Prime Park, Natick, Ma. 01760

