

ZMACS EDITOR REFERENCE

MANUAL REVISION HISTORY

Zmacs Reference (2243192-0001 *A)

Original Issue June 1985
Revision A June 1987

© 1987, Texas Instruments Incorporated. All Rights Reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Texas Instruments Incorporated.

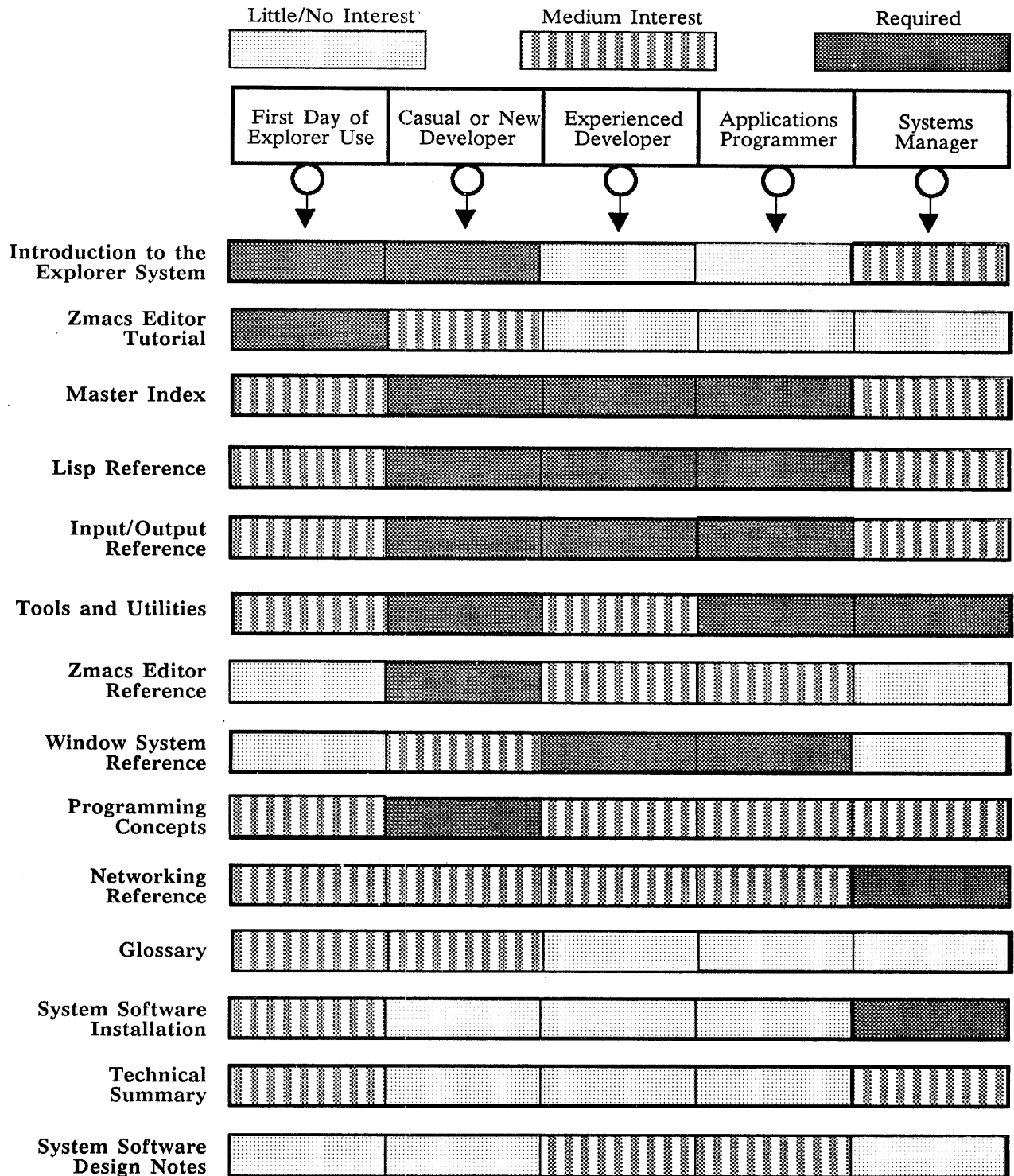
The system-defined windows shown in this manual are examples of the software as this manual goes into production. Later changes in the software may cause the windows on your system to be different from those in the manual.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

Texas Instruments Incorporated
ATTN: Data Systems Group, M/S 2151
P.O. Box 2909
Austin, Texas 78769-2909

THE EXPLORER™ SYSTEM SOFTWARE MANUALS



THE EXPLORER™ SYSTEM SOFTWARE MANUALS

Mastering the Explorer Environment	Explorer Technical Summary	2243189-0001
	Introduction to the Explorer System	2243190-0001
	Explorer Zmacs Editor Tutorial	2243191-0001
	Explorer Glossary	2243134-0001
	Explorer Networking Reference	2243206-0001
	Explorer Diagnostics	2533554-0001
	Explorer Master Index to Software Manuals	2243198-0001
Explorer System Software Installation Guide	2243205-0001	

Programming With the Explorer	Explorer Programming Concepts	2549830-0001
	Explorer Lisp Reference	2243201-0001
	Explorer Input/Output Reference.	2549281-0001
	Explorer Zmacs Editor Reference	2243192-0001
	Explorer Tools and Utilities	2549831-0001
	Explorer Window System Reference	2243200-0001

Explorer Options	Explorer Natural Language Menu System User's Guide	2243202-0001
	Explorer Relational Table Management System User's Guide	2243203-0001
	Explorer Grasper User's Guide	2243135-0001
	Explorer Prolog User's Guide	2537248-0001
	Programming in Prolog, by Clocksin and Mellish	2537157-0001
	Explorer Color Graphics User's Guide	2537157-0001
	Explorer TCP/IP User's Guide	2537150-0001
	Explorer LX™ User's Guide	2537225-0001
	Explorer LX System Installation	2537227-0001
	Explorer NFS™ User's Guide	2546890-0001
	Explorer DECnet™ User's Guide	2537223-0001
	Personal Consultant™ Plus Explorer	2537259-0001

System Software Internals	Explorer System Software Design Notes	2243208-0001
	Release Information, Explorer System Software	2549844-0001

Explorer and NuBus are trademarks of Texas Instruments Incorporated.
Explorer LX is a trademark of Texas Instruments Incorporated.
NFS is a trademark of Sun Microsystems, Inc.
DECnet is a trademark of Digital Equipment Corporation.
Personal Consultant is a trademark of Texas Instruments Incorporated.

THE EXPLORER™ SYSTEM HARDWARE MANUALS

System Level Publications	Explorer 7-Slot System Installation	2243140-0001
	Explorer System Field Maintenance	2243141-0001
	Explorer System Field Maintenance Documentation Kit	2243222-0001
	Explorer System Field Maintenance Supplement	2537183-0001
	Explorer System Field Maintenance Supplement Documentation Kit	2549278-0001
	Explorer NuBus™ System Architecture General Description	2537171-0001

System Enclosure Equipment Publications	Explorer 7-Slot System Enclosure General Description	2243143-0001
	Explorer Memory General Description (8-megabytes)	2533592-0001
	Explorer 32-Megabyte Memory General Description	2537185-0001
	Explorer Processor General Description	2243144-0001
	68020-Based Processor General Description	2537240-0001
	Explorer II Processor and Auxiliary Processor Options General Description	2537187-0001
	Explorer System Interface General Description	2243145-0001
	Explorer NuBus Peripheral Interface General Description (NUPI board)	2243146-0001

Display Terminal Publications	Explorer Display Unit General Description	2243151-0001
	CRT Data Display Service Manual, Panasonic code number FTD85055057C	2537139-0001
	Model 924 Video Display Terminal User's Guide	2544365-0001

143-Megabyte Disk/Tape Enclosure Publications	Explorer Mass Storage Enclosure General Description	2243148-0001
	Explorer Winchester Disk Formatter (ADAPTEC) Supplement to Explorer Mass Storage Enclosure General Description	2243149-0001
	Explorer Winchester Disk Drive (Maxtor) Supplement to Explorer Mass Storage Enclosure General Description	2243150-0001
	Explorer Cartridge Tape Drive (Cipher) Supplement to Explorer Mass Storage Enclosure General Description	2243166-0001
	Explorer Cable Interconnect Board (2236120-0001) Supplement to Explorer Mass Storage Enclosure General Description	2243177-0001

143-Megabyte Disk Drive Vendor Publications	XT-1000 Service Manual, 5 1/4-inch Fixed Disk Drive, Maxtor Corporation, part number 20005 (5 1/4-inch Winchester disk drive, 112 megabytes)	2249999-0001
	ACB-5500 Winchester Disk Controller User's Manual, Adaptec, Inc., (formatter for the 5 1/4-inch Winchester disk drive)	2249933-0001

1/4-Inch Tape Drive Vendor Publications	Series 540 Cartridge Tape Drive Product Description, Cipher Data Products, Inc., Bulletin Number 01-311-0284-1K (1/4-inch tape drive) 2249997-0001 MT01 Tape Controller Technical Manual, Emulex Corporation, part number MT0151001 (formatter for the 1/4-inch tape drive) 2243182-0001
--	---

182-Megabyte Disk/Tape Enclosure MSU II Publications	Mass Storage Unit (MSU II) General Description 2537197-0001
---	--

182-Megabyte Disk Drive Vendor Publications	Control Data® WREN™ III Disk Drive OEM Manual, part number 77738216, Magnetic Peripherals, Inc., a Control Data Company 2546867-0001
--	--

515-Megabyte Mass Storage Subsystem Publications	SMD/515-Megabyte Mass Storage Subsystem General Description (includes SMD/SCSI controller and 515-megabyte disk drive enclosure) 2537244-0001
---	---

515-Megabyte Disk Drive Vendor Publications	515-Megabyte Disk Drive Documentation Master Kit (Volumes 1, 2, and 3), Control Data Corporation 2246129-0002 Volume 1, General Description, Operation, Installation and Checkout, and Part Data 2246125-0004 Volume 2, Theory, General Maintenance, Trouble Analysis, Electrical Checks, and Repair Information 2246125-0005 Volume 3, Diagrams 2246125-0006
--	---

1/2-Inch Tape Drive Publications	MT3201 1/2-Inch Tape Drive General Description 2537246-0001
---	--

1/2-Inch Tape Drive Vendor Publications	Cipher CacheTape® Documentation Manual Kit (Volumes 1 and 2 With SCSI Addendum and, Logic Diagram), Cipher Data products 2246130-0001 1/2-Inch Tape Drive Operation and Maintenance (Volume 1), Cipher Data Products 2246126-0001 1/2-Inch Tape Drive Theory of Operation (Volume 2), Cipher Data Products 2246126-0002 SCSI Addendum With Logic Diagram, Cipher Data Products 2246126-0003
--	---

Control Data is a registered trademark of Control Data Corporation.
WREN is a trademark of Control Data Corporation.
CacheTape is a registered trademark of Cipher Data Products, Inc.

Printer Publications	Model 810 Printer Installation and Operation Manual	2311356-9701
	Omni 800™ Electronic Data Terminals Maintenance Manual for Model 810 Printers	0994386-9701
	Model 850 RO Printer User's Manual	2219890-0001
	Model 850 RO Printer Maintenance Manual	2219896-0001
	Model 850 XL Printer User's Manual	2243250-0001
	Model 850 XL Printer Quick Reference Guide	2243249-0001
	Model 855 Printer Operator's Manual	2225911-0001
	Model 855 Printer Technical Reference Manual	2232822-0001
	Model 855 Printer Maintenance Manual	2225914-0001
	Model 860 XL Printer User's Manual	2239401-0001
	Model 860 XL Printer Maintenance Manual	2239427-0001
	Model 860 XI Printer Quick Reference Guide	2239402-0001
	Model 860/859 Printer Technical Reference Manual	2239407-0001
	Model 865 Printer Operator's Manual	2239405-0001
	Model 865 Printer Maintenance Manual	2239428-0001
	Model 880 Printer User's Manual	2222627-0001
	Model 880 Printer Maintenance Manual	2222628-0001
	OmniLaser™ 2015 Page Printer Operator's Manual	2539178-0001
	OmniLaser 2015 Page Printer Technical Reference	2539179-0001
	OmniLaser 2015 Page Printer Maintenance Manual	2539180-0001
	OmniLaser 2108 Page Printer Operator's Manual	2539348-0001
	OmniLaser 2108 Page Printer Technical Reference	2539349-0001
	OmniLaser 2108 Page Printer Maintenance Manual	2539350-0001
	OmniLaser 2115 Page Printer Operator's Manual	2539344-0001
	OmniLaser 2115 Page Printer Technical Reference	2539345-0001
	OmniLaser 2115 Page Printer Maintenance Manual	2539356-0001

Communications Publications	990 Family Communications Systems Field Reference	2276579-9701
	EI990 Ethernet® Interface Installation and Operation	2234392-9701
	Explorer NuBus Ethernet Controller General Description	2243161-0001
	Communications Carrier Board and Options General Description	2537242-0001

Omni 800 is a trademark of Texas Instruments Incorporated.
OmniLaser is a trademark of Texas Instruments Incorporated.
Ethernet is a registered trademark of Xerox Corporation.

CONTENTS

Section	Title
	About This Manual
1	Zmacs Overview
2	Zmacs Operations
3	Command Groups
4	Customizing Zmacs

Paragraph	Title	Page
About This Manual		
	Introduction	xix
	Contents	xix
	Executing Commands	xix
	Using Keystroke Sequences	xix
	Typing Command Names	xx
	Using the Mouse	xx
	Lisp Language Notation	xxi
<hr/>		
1	Zmacs Overview	
1.1	Introduction	1-1
1.2	Text Editing	1-2
1.3	Program Development	1-2
1.4	Getting Started and Help	1-4
1.5	Text Storage Facilities	1-4
1.5.1	Files, Buffers, and Directories	1-5
1.5.2	Kill History	1-6
1.5.3	Point PDL	1-6
1.5.4	Registers	1-6
1.5.5	Keyboard Macros	1-6
1.6	Searching and Replacing	1-7
1.7	Customization	1-7
<hr/>		
2	Zmacs Operations	
2.1	Introduction	2-1
2.2	Entering Zmacs	2-2
2.3	Exiting Zmacs	2-3
2.4	Zmacs Screen	2-3
2.4.1	Editor Buffer Window	2-3
2.4.2	Mode Line Window	2-4
2.4.3	Mouse Documentation Window	2-5
2.4.4	System Status Line	2-5
2.4.5	Scroll Bar	2-5
2.4.6	Timeout Window	2-5
2.5	Executing Commands	2-6
2.6	Scrolling	2-7
2.7	Splitting the Screen	2-7
2.8	Menus	2-9
2.8.1	Types of Menus	2-10
2.8.2	System Menu	2-10
2.8.3	Top-Level Editor Menu	2-11
2.8.4	Zmacs Suggestions Menus	2-11

Paragraph	Title	Page
2.9	Help Facilities	2-12
2.9.1	Help Menu Commands	2-12
2.9.2	Help Commands for Lisp Code	2-13
2.9.3	Aborting	2-14
2.9.4	Troubleshooting	2-14
2.10	Files, Buffers, and Directories	2-15
2.10.1	Pathname Structure	2-16
2.10.2	File Types	2-18
2.10.3	Mode Line Information	2-18
2.10.4	Lisp Code in Buffers	2-19
2.10.5	Creating Directories	2-20
2.10.6	Creating Buffers	2-20
2.10.7	Listing and Editing Buffers	2-20
2.10.8	Finding a File for Editing	2-22
2.10.9	Saving and Writing Files	2-22
2.10.10	Editing a Directory	2-22
2.11	Cursor Movement	2-23
2.11.1	General Rules for Cursor Movement	2-24
2.11.2	Cursor Movement Commands Grouped by Quantity	2-26
2.11.3	Storing Cursor Locations	2-26
2.11.3.1	Storing Cursor Locations on the Point PDL	2-27
2.11.3.2	Storing Cursor Locations in Registers	2-29
2.12	Setting Modes and Buffer Attributes	2-30
2.12.1	Setting Modes	2-30
2.12.2	Setting Buffer Attributes	2-32
2.13	Deleting and Moving Text	2-34
2.13.1	Exchanging Text	2-36
2.13.2	Marking Text	2-36
2.13.3	Deleting and Killing Text	2-39
2.13.4	Retrieving (Yanking) Text	2-41
2.13.5	Storing Text in Registers	2-43

3

Command Groups

3.1	Overview of Commands	3-1
3.1.1	Executing Commands	3-2
3.1.2	Prefix Command	3-3
3.1.3	Minibuffer Commands	3-3
3.1.4	Numeric Arguments	3-5
3.2	Buffer Commands	3-7
3.2.1	Capture Into Buffer Commands	3-7
3.2.2	Insert Buffer Commands	3-8
3.2.3	Kill and Save Buffer Commands	3-8
3.2.4	List and Edit Buffer Commands	3-10
3.2.5	List and Edit Changed Definition Commands	3-15
3.2.6	Miscellaneous Buffer Commands	3-18
3.2.7	Print Buffer Commands	3-19
3.2.8	Revert Buffer Command	3-20
3.2.9	Select Buffer Commands	3-20
3.3	Compile and Evaluate Commands	3-21
3.3.1	Evaluate Minibuffer Command	3-23
3.3.2	Evaluate Into Buffer Commands	3-23
3.3.3	Compile or Evaluate Region or Definition Commands	3-23

Paragraph	Title	Page
3.3.4	Compile or Evaluate Changed Definition Commands	3-25
3.3.5	Compile or Evaluate Buffer Commands	3-26
3.3.6	Compile File Command	3-27
3.3.7	Update Xld Command	3-28
3.3.8	Compile and Load Commands	3-28
3.3.9	Compiler Warning Commands	3-29
3.3.10	Disassemble Commands	3-32
3.4	Cursor Movement Commands	3-33
3.4.1	General Cursor Movement Commands	3-33
3.4.2	Lisp Cursor Movement Commands	3-35
3.4.3	Text Cursor Movement Commands	3-40
3.4.4	Saving Cursor Locations (Point) Commands	3-41
3.4.5	Various Quantities Command	3-45
3.5	Customization Commands	3-47
3.5.1	Install Command on Key Commands	3-47
3.5.2	Keyboard Macro Commands	3-48
3.5.3	Variable Commands	3-52
3.5.4	Word Abbreviation Commands	3-54
3.6	Deleting and Moving Text Commands	3-59
3.6.1	Delete Commands	3-59
3.6.2	Kill Commands	3-61
3.6.3	Mark Commands	3-65
3.6.4	Register Commands	3-68
3.6.5	Yank (Retrieve) Commands	3-72
3.7	Directory Commands	3-75
3.7.1	Create, List, and Clean Directory Commands	3-75
3.7.2	Edit a Directory (Dired) Commands	3-77
3.7.2.1	Dired Enter Commands	3-80
3.7.2.2	Dired Documentation Command	3-80
3.7.2.3	Dired Abort Command	3-80
3.7.2.4	Cursor Movement Commands	3-80
3.7.2.5	Numeric Arguments	3-81
3.7.2.6	Dired Edit Commands	3-81
3.7.2.7	Dired Find File Command	3-82
3.7.2.8	Dired View File Command	3-82
3.7.2.9	Dired Delete Command	3-82
3.7.2.10	Dired Print File Command	3-83
3.7.2.11	Dired Apply Function Command	3-83
3.7.2.12	Dired Undelete Commands	3-83
3.7.2.13	Dired Hog Commands	3-84
3.7.2.14	Dired Execute Commands	3-85
3.7.2.15	Dired Rename Command	3-86
3.7.2.16	Dired Copy Command	3-86
3.7.2.17	Dired Compile and Load File Commands	3-86
3.7.2.18	Dired Subdirectory Command	3-87
3.7.2.19	Dired Next Undumped Command	3-88
3.7.2.20	Dired Flag Commands	3-88
3.7.2.21	Dired Print File Attributes Command	3-88
3.7.2.22	Dired Change File Properties Command	3-89
3.7.2.23	Dired SRCCOM Command	3-90
3.7.2.24	Dired Sort Commands	3-90
3.7.2.25	Dired Mouse Commands	3-92

Paragraph	Title	Page
3.8	File Commands	3-93
3.8.1	Change File Properties Command	3-93
3.8.2	Copy File Commands	3-94
3.8.3	Delete File Commands	3-95
3.8.4	Find and View File Commands	3-96
3.8.5	Insert File Commands	3-97
3.8.6	Print File Command	3-98
3.8.7	Rename File Commands	3-98
3.8.8	Save and Write File Commands	3-99
3.9	Font Commands	3-101
3.10	Help, Documentation, and Undo Commands	3-105
3.10.1	Abort Commands	3-105
3.10.2	Command and Key Help Commands	3-106
3.10.3	Function and Variable Help Commands	3-109
3.10.4	Help Menu Commands	3-113
3.10.5	List Keystroke History Command	3-114
3.10.6	Minibuffer Help Commands	3-114
3.10.7	Where Am I Command	3-115
3.10.8	Symbol Help Command	3-115
3.10.9	Teach Zmacs Command	3-115
3.10.10	Undo Commands	3-115
3.11	Lisp Programming Commands	3-117
3.11.1	Break Command	3-117
3.11.2	Caller Commands	3-117
3.11.3	Edit or Find Source Commands	3-119
3.11.4	Flavor Commands	3-120
3.11.5	Set Package Command	3-122
3.11.6	Patch Commands	3-124
3.11.7	Possibility Commands	3-126
3.11.8	Source Compare Commands	3-128
3.11.9	Trace Command	3-132
3.12	Lisp Syntax Commands	3-133
3.12.1	Comment Commands	3-133
3.12.2	Grind (Pretty Print) Commands	3-136
3.12.3	Macro Expansion Commands	3-136
3.12.4	Parentheses Commands	3-137
3.13	Miscellaneous Commands	3-141
3.14	Mode and Buffer Attribute Commands	3-143
3.14.1	Major Mode Commands	3-143
3.14.2	Minor Mode Commands	3-144
3.14.3	Buffer Attribute Commands	3-147
3.15	Mouse Commands	3-151
3.16	Print Commands	3-153
3.17	Scroll Commands	3-155
3.18	Search and Replace Commands	3-159
3.18.1	Search Commands	3-159
3.18.1.1	String Search Commands	3-159
3.18.1.2	Incremental Search Commands	3-161
3.18.1.3	Extended Search Commands	3-163
3.18.1.4	Miscellaneous Search Commands	3-166
3.18.2	Replace Commands	3-168
3.18.3	Tag Commands	3-172

Paragraph	Title	Page
3.19	Text Format Commands	3-179
3.19.1	Exchange Commands	3-179
3.19.2	Fill Commands	3-182
3.19.3	Lowercase and Uppercase Commands	3-184
3.19.4	Miscellaneous Text Format Commands	3-185
3.19.5	Sort Commands	3-187
3.19.6	Tab and Indentation Commands	3-188
3.19.6.1	Indentation Commands	3-189
3.19.6.2	Tab Commands	3-194
3.19.7	Just One Space Command	3-196
3.20	Window Commands	3-197

4 **Customizing Zmacs**

4.1	Introduction	4-1
4.2	Changing User Variables	4-1
4.2.1	Listing and Changing User Variables	4-2
4.2.2	Examples of Changing User Variables	4-2
4.3	Customizing Keys	4-3
4.4	Login Init File	4-3
4.5	Writing Your Own Commands	4-5
4.6	Standalone Editor	4-6

Index

	Figure	Title	Page
Figures	2-1	Zmacs Screen	2-4
	2-2	Typeout Window	2-6
	2-3	Split Screen Display (First Screen)	2-8
	2-4	Split Screen Display (Second Screen)	2-8
	2-5	Editor Menu	2-11
	2-6	Show Point PDL Command	2-28
	2-7	Example Attribute List	2-34
	2-8	Show Kill History Display	2-41
	3-1	Kill or Save Buffers Menu	3-9
	3-2	Sample Buffer History	3-13
	3-3	List Buffers Menu	3-14
	3-4	List Changed Definitions Command	3-16
	3-5	List Buffer Changed Definitions Command	3-17
	3-6	List Sections Command	3-17
	3-7	Show Point PDL Command	3-44
	3-8	Variable Apropos Command (HELP V)	3-53
	3-9	List Registers Command	3-69
	3-10	Show All Registers Command	3-69
	3-11	Show Saved Positions Command	3-70
	3-12	View Register Command	3-71
	3-13	Example Dired Listing	3-77
	3-14	Dired Menu	3-78
	3-15	Dired Change File Properties Menu	3-89
	3-16	Change File Properties Menu	3-94
	3-17	Apropos Command (HELP A)	3-107
	3-18	Self Document Command (HELP C)	3-108
	3-19	Where Is Command (HELP W)	3-108
	3-20	Function Apropos Command	3-110
	3-21	Function Apropos Possibilities Buffer	3-111
3-22	Variable Apropos Command (HELP V)	3-112	
3-23	Undo Command (HELP U)	3-116	
3-24	Sample Definitions Buffer	3-127	
3-25	Source Compare Merge Command	3-130	
3-26	Trace Command Menu	3-132	
3-27	Tab-Stop Buffer	3-194	
3-28	Split Screen Display (First Screen)	3-200	
3-29	Split Screen Display (Second Screen)	3-200	

	Table	Title	Page
Table	3-1	File Properties	3-89

ABOUT THIS MANUAL

Introduction

The *Explorer Zmacs Editor Reference* describes the Zmacs editor, a command-driven, window-oriented editor that provides a wide range of facilities for editing text, developing Lisp programs (editing source code), and manipulating files. The manual tells you how to operate Zmacs and provides reference material on all the Zmacs commands. The manual is intended for anyone using the Explorer system. It assumes you have read the *Explorer Technical Summary* and are familiar with the basic operations described in the *Introduction to the Explorer System*.

Contents

This manual contains an index and the following sections:

Section 1: Zmacs Overview — Presents an overview of the features available on Zmacs, which range from compiling and evaluating Lisp code within Zmacs, to elaborate search and replace operations, to online help on almost any operation you are performing.

Section 2: Zmacs Operations — Describes how to perform basic operations on Zmacs, such as entering and exiting Zmacs, creating directories and files, moving the cursor, and deleting and moving text.

Section 3: Command Groups — Presents all the Zmacs commands in functional groups to explain how they work together and to help you easily find a particular command.

Section 4: Customizing Zmacs — Tells how you can easily change some of the ways Zmacs operates.

Executing Commands

You can execute Zmacs commands three ways: by using keystroke sequences, by typing the command name, and by using the mouse.

Using Keystroke Sequences

You can execute many of the commands in Zmacs with a combination or sequence of keystrokes. In this manual, hyphens connect the names of keys that you should press simultaneously (*chord*). Spaces separate the names of keys that you should press one after the other. The following table illustrates this manual's conventions for describing keystroke sequences.

Keystroke Sequence	Description
SYSTEM E	Press the SYSTEM key and release it, then press the E key and release it.
META-X	Hold the META key and press the X key.
CTRL-X CTRL-F	Hold the CTRL key and press the X key, release the X key, and then press the F key. Alternatively, press CTRL-X, release both keys, and press CTRL-F.

**Typing
Command Names**

Many Zmacs commands do not have keystrokes assigned to them. To execute these commands, you press META-X, type the name of the command, and then press RETURN.

META-X Find File
RETURN Hold the META key and press the X key, release the keys, type the words `find file` (separated by a space), and then press the RETURN key.

Using the Mouse

The optical mouse features three buttons that enable you to perform operations from the mouse. The mouse documentation window (the window in reverse video at the bottom of the screen) tells you what operations you can perform with the mouse. The options change according to the operation you are performing. The following table describes the abbreviations the mouse documentation window uses to describe using the mouse buttons. (Pressing and releasing a button is called *clicking*.)

Abbreviation	Action
L	Click the left button (press the left button once and release).
M	Click the middle button (press the middle button once and release).
R	Click the right button (press the right button once and release).
L2 M2 R2	Click the specified button twice quickly. (Press the button, release it, then press it again quickly.) This action is called <i>double clicking</i> *.
LHOLD MHOLD RHOLD	Press the specified button and hold it down.

* If you double click too fast, the system sees only one click. If you double click too slowly, the system sees two single clicks. You can use an alternative method to prevent such misinterpretations: press and hold the CTRL key while you click the specified button one time.

Lisp Language Notation

The Lisp language notational convention helps you distinguish Lisp functions and arguments from user-defined symbols. The following table shows the three fonts used in this manual to denote Lisp code:

Typeface	Meaning
boldface	System-defined words and symbols, including names of functions, macros, flavors, methods, variables, keywords, and so on—any word or symbol that appears in the system source code.
<i>italics</i>	Example names and arguments to functions, such as a value or parameter you specify. An item in italics can be replaced by any value you choose to substitute. (Italics are also used for emphasis and to introduce new terms.)
monowidth	Examples of program code and output. System-defined words shown in examples are also in this font.

For example, this sentence contains the word **setf** in boldface because **setf** is defined by the system.

Some function and method names are very long—for example, **get-ucode-version-of-band**. Within the text, long function names may be split over two lines because of typographical constraints. When you code the function name **get-ucode-version-of-band**, however, you should not split it or include any spaces within it.

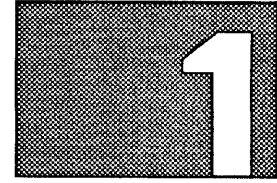
Within manual text, each example of actual Lisp code is shown in monowidth font. For instance:

```
(setf x 1 y 2) => 2
(+ x y) => 3
```

The form `(setf x 1 y 2)` sets the variables `x` and `y` to integer values; then the form `(+ x y)` adds them together.

In this example of Lisp code with its explanation, **setf** appears in the monowidth font because it is part of a specific example.

For more detailed information about Lisp syntax descriptions, see Section 1, Introduction, of the *Explorer Lisp Reference* manual.



ZMACS OVERVIEW

Introduction

1.1 The Zmacs editor is a command-driven, window-oriented editor that you can use for text editing, program development (source code editing), and file manipulation. Zmacs is descended from Emacs, a powerful text editor developed at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Zmacs provides these features:

- Text editing capabilities — Zmacs provides commands that make it easy for you to edit text. These commands allow you to perform operations on different units of text: character, word, sentence, paragraph, and so on. Cursor movement commands quickly move you to whatever location you want. You can easily move and copy blocks of text.
- Program development — Closely tied to the other utilities on the system, Zmacs lets you compile and evaluate Lisp code and helps you find source code for editing. Zmacs also enables you to find functions and variables. When you are writing Lisp code, Zmacs helps you with balancing parentheses and proper indentation. The commands available allow you to perform operations such as cursor movement on different units of Lisp code: symbols, lists, and so on.

The integrated environment of the Explorer system, which allows you to use the interpreter and the compiler from inside the editor, adds to the power of Zmacs. Also, you can easily go to another program such as the Inspector by using the System menu (which is available from Zmacs and everywhere else).

- Getting started and help — The *Explorer Zmacs Editor Tutorial* gives the first-time user valuable hands-on experience. It covers a wide range of basic commands. This reference manual discusses basic operations in more detail.

Online help is available for almost every situation. Pressing the HELP key usually displays information about whatever operation you are performing. Other help commands assist you in finding a command name, a keystroke, or even a variable name.

The Zmacs Suggestions menus display the commands in functional groups. They make it easy for you to find and execute a particular command. Most people like to use the menus when they first start because the menus are easy to use and quickly help you learn your way around.

- Text storage facilities — Zmacs contains a large repertoire of commands that deal with files and directories. It also contains temporary storage locations for saving blocks of text and cursor locations.
- Searching and replacing — Zmacs provides commands that perform a wide range of search and replace operations. You can search and replace forward or backward.

- Customization — You can change how Zmacs operates and tailor it to suit your individual needs. Zmacs contains facilities to help you easily change some aspects of its workings.

Text Editing

1.2 Zmacs provides extensive text editing capabilities. Zmacs contains a mode, called *Text mode*, that is specifically designed for editing text. A mode changes how some commands operate.

Zmacs contains commands to manipulate text at the following levels: character, word, line, sentence, paragraph, region, and buffer/file. A *region* is a block of text that you define. Some of the operations you can perform include the following:

- Change text to uppercase or lowercase.
- Delete text.
- Fill text. *Fill* means to put as much text as possible on the current line without exceeding the right margin.
- Justify text. *Justify* means to make the text align on the right margin.
- Move the cursor to a different location.
- Set tabs where you want them.
- Transpose text.

Program Development

1.3 Zmacs provides extensive program development aids for Lisp code. It is closely tied to the compiler and interpreter, so you can compile and evaluate from within the editor. Zmacs also contains two modes specifically designed for Lisp code: *Common Lisp mode* for Common Lisp code and *Zetalisp mode* for Zetalisp code.

Zmacs provides commands to manipulate Lisp code on the following levels:

- Symbolic expression
- Definition
- Region

When you edit a file of function definitions, each definition is a *section* within the buffer that you are editing. After the opening parenthesis in the first column, a section can contain any defining construct that begins with *def*, such as *defvar*, *defflower*, and *defun*. The section scheme allows you to locate definitions quickly. It also allows you to perform operations such as compiling only the changed sections (or definitions) in a buffer. (Buffers and files are explained later.)

Zmacs helps you balance parentheses by providing automatic display of matching parentheses. When you finish typing the closing parenthesis for an expression, Zmacs flashes the corresponding opening parenthesis. If the

cursor is on the opening parenthesis, Zmacs flashes the corresponding closing parenthesis if it exists.

Zmacs also provides automatic indentation. It correctly indents the next line for you. It also provides commands to perform more elaborate indentation, such as indenting every line in a block of code that you define as a region.

Among the other program development operations with Lisp code are the following:

- Finding information about Lisp code. Lisp finds information about code in real time, whereas other languages stop at mere cross-references. You can use the mouse cursor to point to a symbol in a Lisp expression, and then enter various Zmacs commands to provide information about the symbol and find its source. Examples of operations you can perform with Zmacs commands are as follows:
 - Find the source code for a function.
 - List and edit callers.
 - Edit flavors and methods.
 - Display the expansion of a Lisp macro.
 - Manipulate variables. Zmacs helps you find variables, prints documentation about a variable, and allows you to change the values of variables.
 - List the arguments for a function.
 - Provide documentation about a function.
 - Trace the execution of a function.
- Improving code appearance and pretty printing. Zmacs helps you with the appearance and indentation of Lisp code. For example, Zmacs enables you to perform these tasks:
 - Change Lisp code to uppercase, lowercase, or initial capitals.
 - Grind and indent Lisp code. *Grind* means to indent Lisp code correctly (or to pretty print it). *Grind* works on a whole Lisp expression, whereas *indent* usually works on one or more lines that you specify or take as the default.
- Understanding program structure. Zmacs provides commands to manipulate the syntax of a Lisp program. For example, Zmacs helps you to perform these tasks:
 - Manipulate comments. You can move the cursor by comment lines, indent comments, and start and end comments.
 - Manipulate parentheses. You can move the cursor from one parenthesis to the next, display the start of a list, and insert matching parentheses.

Getting Started and Help

1.4 The *Explorer Zmacs Editor Tutorial* gives the first-time user valuable hands-on experience with Zmacs. It teaches many of the basic commands and operations that Zmacs can perform.

Section 2 of this reference manual describes the basic operations of Zmacs in more detail. You can use Section 3 for reference. It describes all the Zmacs commands in functional groups.

Zmacs contains many help facilities. Help is available for almost every situation. Usually, you can press the HELP key to display information about the operation you are performing. Among the help facilities available are the following:

- A Help menu that provides a wide range of help operations
- Documentation on an individual command
- Documentation on a key sequence
- Documentation on a function
- Documentation on a variable
- A listing of the last 60 keystrokes you typed
- A command to undo your recent editing changes

The Zmacs Suggestions menus help you get started using Zmacs by displaying the Zmacs commands in functional groups. You can see these menu names when you first enter Zmacs. These menus teach you the use and meaning of various commands and make you aware of all the commands in a given context. You can *select* a menu item with the mouse. To select a menu item, you position the mouse cursor on the item until it is highlighted with a box and then you click left. Clicking on an item either displays more menu names, the individual commands in the group, or both. If you select an individual command, the command is executed.

To turn on the Suggestions menus, you select the item Suggestions Menus On in the command line in the Lisp Listener.

Text Storage Facilities

1.5 Zmacs can access the following text storage facilities. You can use these depending on your particular needs.

- Files — Stored on disk.
- Buffers — Memory-resident versions of files that Zmacs uses for editing.
- Directories — Contain both files and other directories.
- Kill history — Used to move and copy text.
- Point PDL — Stores cursor locations. (PDL stands for push-down list.)

- Registers — Store cursor locations and/or text.
- Keyboard macros — Store a series of commands that you can later call as one command.

Files and directories are part of the Explorer file system. Buffers, registers, keyboard macros, the kill history, and the point PDL are part of Zmacs.

Files, Buffers, and Directories

1.5.1 Zmacs provides a wide range of file, buffer, and directory operations. Zmacs makes a distinction between files and buffers. *Files* are stored on disk. Zmacs does not edit files. Whenever you want to edit a file, Zmacs makes a temporary copy in a *buffer*. You work with the copy in the buffer. When you save a buffer, it is written to a file on disk.

When you write a buffer to disk, it does not overwrite the previous copy of the file. It has the same filename but the next higher *version number*.

All files and buffers are stored in directories, which form a tree structure. A *directory* is a file that contains other files or subdirectories. A *subdirectory* is a directory within another directory. A subdirectory can also contain files and subdirectories. Zmacs provides a directory editor, *Dired*, that allows you to perform a wide range of operations on a directory.

Some of the file, buffer, and directory operations available are as follows:

- Directory housekeeping. You can perform all your directory housekeeping from *Dired*. You can delete, copy, rename, print, and edit files. However, you do not immediately delete files. You mark them for deletion, and you can remove the deletion mark. When you finish with *Dired*, you are asked whether you want to delete the files that are marked for deletion.
- Editing and viewing. Zmacs provides several different operations for editing and viewing files and buffers:
 - Find — Brings a file into a buffer for editing.
 - Select — Allows you to switch between buffers.
 - Edit — Allows you to edit a buffer.
 - View — Allows you to view the contents of a file or buffer, but you cannot edit the file or buffer.
- Saving and writing files. Zmacs provides two commands that copy a buffer to a file. The Write File command lets you specify the filename. The Save command automatically gives you the default filename.
- Compiling and loading. Zmacs lets you compile files and buffers from within Zmacs. The integrated environment of the Explorer system allows Zmacs to use the compiler. You can also *load* a file. Loading means to bring the contents of the file on *disk* into *memory* so that the functions and so on can be called.
- Moving blocks of text. You can move blocks of text within buffers and between buffers. You can also insert a file into your buffer.

Files and buffers use fonts. A *font* specifies the typeface and size of characters in the file or buffer. The Explorer system contains a wide assortment of fonts that you can use in Zmacs. You can highlight different parts of your text by using different fonts.

Kill History 1.5.2 The *kill history* is a stack of infinite size that allows you to move and copy blocks of text. The operations you perform with the kill history are called kill, save, and yank.

When you *kill* text, it is put on the kill history and removed from your buffer. You can *save* text on the kill history without removing it from your buffer. When you *yank* text, you *copy* an entry from the kill history into your buffer. You can yank all or only parts of the entry. A save followed by a yank is the way Zmacs allows you to copy text. A kill followed by a yank is the way Zmacs allows you to move text.

Usually, the block of text in the kill history is a region that you have marked. When you *mark* a block of text, it is highlighted (underlined or displayed in reverse video) and referred to as a *region*.

Point PDL 1.5.3 The point PDL allows you to store point (or cursor) locations. You can use the point PDL to return to previous cursor locations. Point refers to the previous location of the keyboard cursor. *PDL* stands for push-down list, which basically means that entries are added to and removed from the top of the list. (In modern terminology, a PDL is a stack.)

Point is actually the position in the buffer represented by the left-hand edge of the keyboard cursor. Point is never on a character as the cursor is; point is always between characters. For example, if the cursor is on the *e* in the word *me*, point is between the *m* and the *e*.

Registers 1.5.4 Zmacs contains storage areas called *registers*. The following list describes what you can store in a register:

- A block of text
- A cursor location
- Both a block of text and a cursor location

These registers allow you to store text (or code) in much the same way as the kill history does. The difference is that text put in a register stays there, while the top of the kill history (the part from which you yank) often changes.

Keyboard Macros 1.5.5 A *keyboard macro* is a sequence of commands stored for use as a group. You press several keys that perform commands and make that a repeatable sequence. You can name the keyboard macro or store it on a keystroke for easy execution. This is one of the customization features that Zmacs offers.

Searching and Replacing

1.6 Zmacs contains a wide variety of search and replace facilities. You can search and replace forward or backward. The following search facilities are available:

- *String search* finds the string that you specify.
- *Incremental search* finds intermediate strings while you are typing a string.
- *Extended search* lets you make complicated searches for combinations of strings by using special characters.
- *Tags tables search* allows you to search through groups of files. The files are treated as one file for the search operation.

The following replace facilities are available:

- *String replace* replaces all occurrences of a given string with another.
- *Query replace* replaces a string and asks you about each occurrence (that is, whether you want to replace it).
- *Multiple query replace* replaces two sets of strings at the same time and asks you about each occurrence.
- *Tags tables replace* allows you to replace strings in a group of files. The files are treated as one file for the replace operation.

Customization

1.7 Zmacs allows you to change various aspects of its workings, and it provides facilities to help you make the changes.

Zmacs contains a set of variables called *user variables* that allow you to customize Zmacs. Zmacs contains commands that help you find and change the values of these variables. For example, when you mark a region, the region is underlined. You can change the underlining to reverse video.

If you find that you frequently use a command that does not have an associated key, you can associate a key with the command. You can also define a keyboard macro to associate a key with a group of commands. Then, that one key executes the entire group of commands.

If Zmacs does not contain the appropriate command for your particular application, you can customize Zmacs by writing your own command. You might want a command to perform a file or buffer operation that Zmacs does not presently supply.

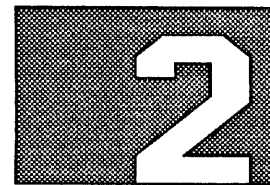
Zmacs contains many minor modes that change some aspects of how Zmacs operates. Text mode, Common Lisp mode, and Zetalisp mode are *major modes*. You always have one and only one major mode active at a time. A *minor mode* is used in conjunction with a major mode. You can specify as many minor modes as you want with any major mode. For example, Zmacs, by default, is in *Insert mode*. When you position the cursor over a character and type another character, the old character is moved to the right and the

new character is inserted. If you specify *Overwrite mode* as a minor mode, the old character is overwritten.

You can create a *login initialization (init) file* to customize Zmacs for you. You set up the file with whatever options you want. The system executes the init file whenever you log in. This init file, which is named LOGIN-INIT.LISP, is in the directory with the same name as your login user ID. That is, if your user ID is JONES, then your init file is JONES; LOGIN-INIT.LISP. (You can also use a compiled version of the file.)

Section 4, Customizing Zmacs, explains customization in detail.

ZMACS OPERATIONS



Introduction

2.1 This section covers the basic operations of Zmacs. It is intended to get you started using Zmacs. It does not describe every detail of the operations, nor does it cover every operation that Zmacs provides. For detailed information, refer to Section 3, which describes all the commands that Zmacs contains. Before using this section, you should be familiar with the *Introduction to the Explorer System*. In addition, you should understand basic operating instructions for the Explorer system.

The following list describes the basic operations that are discussed:

- Entering Zmacs — Tells how to enter Zmacs.
- Exiting Zmacs — Tells how to exit Zmacs.
- Zmacs screen — Describes the Zmacs screen and its various parts.
- Executing commands — Describes the different ways you can execute commands in Zmacs.
- Scrolling — Describes how to scroll in Zmacs.
- Splitting the screen — Explains how to split the screen.
- Menus — Presents an overview of the menus you see in Zmacs.
- Help facilities — Presents an overview of the Zmacs help facilities.
- Files, buffers, and directories — Describes the basic file, buffer, and directory operations, such as creating directories.
- Cursor movement — Describes the cursor movement commands and how to store cursor locations.
- Setting modes and buffer attributes — Tells how to set modes and other buffer attributes.
- Deleting and moving text — Tells how to delete and move text in Zmacs.

NOTE: Unlike some editors, Zmacs is in Insert mode by default. When you position the cursor over a character and type a new character, the old character is moved to the right and the new character is inserted. You can customize Zmacs by putting it in Overwrite mode. Then old characters are overwritten. (Refer to paragraph 2.12, Setting Modes and Buffer Attributes.)

Entering Zmacs

2.2 You can enter Zmacs in several different ways. You can create a new instance of the Zmacs editor, or enter one that already exists. In other words, you can create more than one instance of the editor while you are logged in. However, one instance is sufficient to edit many different buffers. (In this manual, creating an instance of the Zmacs editor is called *creating an editor*.)

You can use any of the following methods to enter the editor:

- **SYSTEM E** — Pressing SYSTEM E creates an editor if one does not already exist. If you have created several editors, SYSTEM E puts you in the most recently used (not recently created) editor. You can cycle through the editors by repeatedly pressing SYSTEM E.
- **SYSTEM CTRL-E** — Pressing SYSTEM CTRL-E creates a new editor, whether one already exists or not.
- **Zmacs Editor option on the System menu** — Selecting the Zmacs Editor option on the System menu creates an editor if one does not already exist. If you have created several editors, selecting Zmacs Editor puts you in the most recently used (not most recently created) editor.

You can also enter the editor from the System menu by selecting Create or Select. The Create option allows you to create a new editor and specify the size and position of the editor window, which otherwise is a full screen. The Select option allows you to select an existing editor. Select gives you a list of the windows currently available, including all the instances of the editor. It shows you the most recently used buffer in each editor. For information on how to use the Create and Select options, refer to the *Introduction to the Explorer System*.

To invoke the System menu when you are in the editor, you double click right.

- **Function calls** — You can also enter the editor by using one of the following function calls:

ed &optional <i>x</i>	Function
----------------------------------	-----------------

The **ed** function is the main function for entering the editor.

The (**ed**) or (**ed nil**) form enters the editor, putting you in the same buffer that you were in the last time you used the editor.

The (**ed t**) form puts you in a new buffer with a generated name (such as *BUFFER-2*).

The (**ed *pathname***) form edits the file specified by *pathname*, which can be an actual pathname or a string.

The (**ed '*foo***) form edits the definition of the *foo* function. It finds a buffer or file containing the source code for *foo* and positions the cursor at the beginning of the code. In general, *foo* can be any function-spec (refer to the *Explorer Lisp Reference*).

The (ed ^zwei:reload) form reinitializes the editor (that is, all instances of the editor). Because it deletes all existing buffers, you should use this form only as a last resort.

dired & optional <i>pathname</i>	Function
This function allows you to edit the directory specified by <i>pathname</i> . If you do not specify a <i>pathname</i> , it defaults to the last file opened.	

Exiting Zmacs

2.3 You can exit Zmacs using any of the following methods:

- END — Pressing END returns you to the previous window you were in. If you were in the Lisp Listener, END returns you to it. However, if you were in another editor, END returns you to this editor.
- System menu — You can exit Zmacs by double clicking right to invoke the System menu and selecting one of the other programs, such as the Lisp Listener.
- SYSTEM HELP — Pressing SYSTEM HELP provides a list of other programs that you can select on the system. You press the SYSTEM key, then the character for the program, such as L for the Lisp Listener (that is, you press SYSTEM L).

Zmacs Screen

2.4 The Zmacs screen contains several related parts, each used for different operations. Figure 2-1 shows the Zmacs screen.

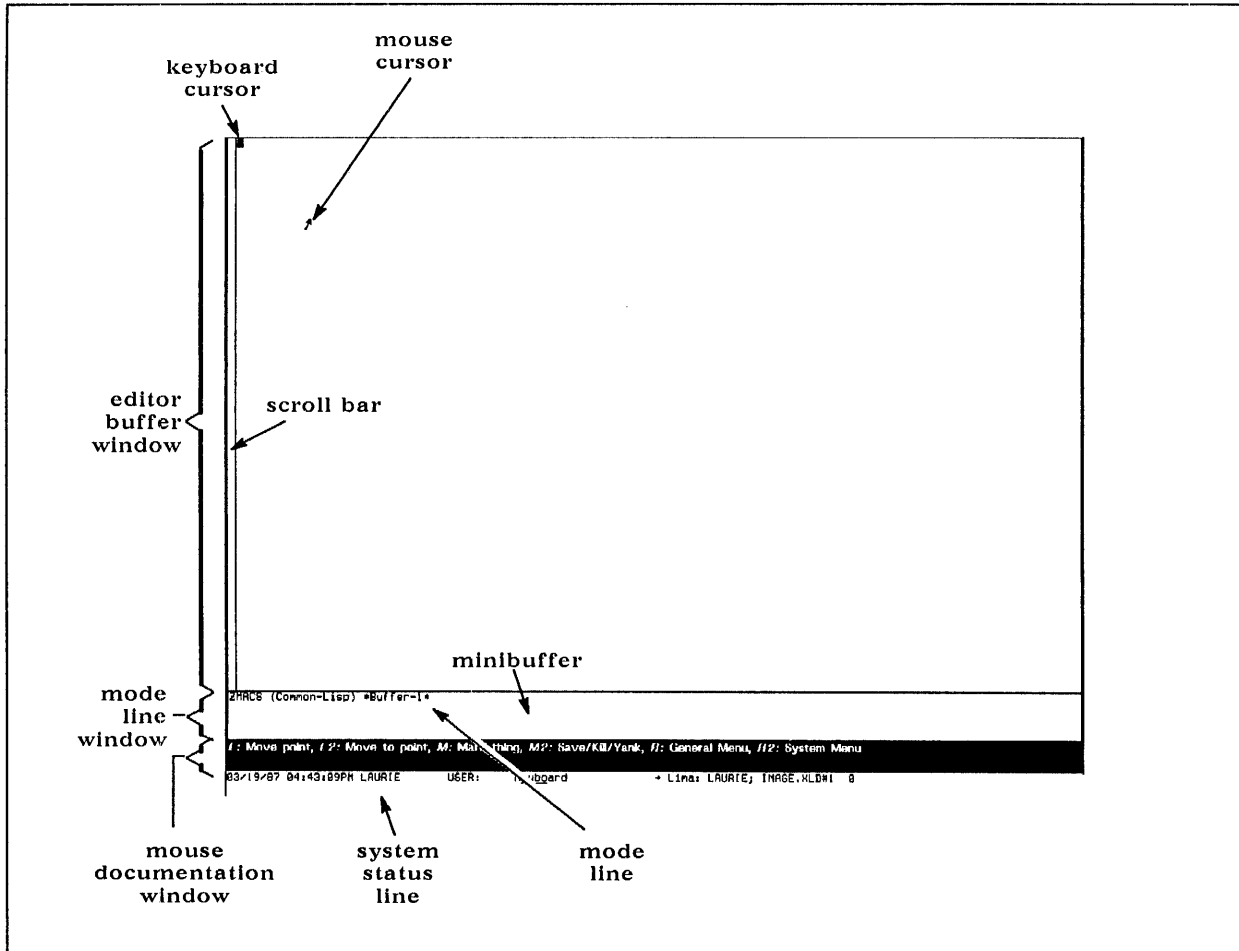
Subsequent paragraphs summarize the purpose of each of the following parts of the Zmacs screen:

- Editor buffer window
- Mode line window
- Mouse documentation window
- System status line
- Scroll bar
- Typeout window

Editor Buffer Window

2.4.1 The editor buffer window is the upper section of the screen, which you use for editing. You can split the screen into two or more editor buffer windows. You can move between multiple editor buffer windows by using either commands or the mouse. You exit the editor buffer window by moving to another editor window or by leaving the editor. For details on splitting the screen, see paragraph 2.7, Splitting the Screen.

Figure 2-1 Zmacs Screen



Mode Line Window

2.4.2 The mode line window contains the mode line and the minibuffer. The *mode line* tells you the following information:

- Program name — The name of the program, which is Zmacs.
- Mode — The mode you are in. Common Lisp mode is the default mode. (Paragraph 2.12, Setting Modes and Buffer Attributes, describes modes.)
- Buffer name — The name of the buffer you are editing. If a buffer is connected to a file (that is, the buffer was read from disk), the version number of the file appears in parentheses beside the pathname. (When you save the contents of a buffer, Zmacs writes the buffer to a file on disk.)
- Font — The name of the current font if fonts have been set for the buffer.
- Size of buffer — If more of the buffer is above or below what you see on the screen, an up arrow or a down arrow is displayed following the name of the buffer.

- **Modified buffer** — If the buffer has been changed since it was read from disk, an asterisk is displayed after the up arrow or down arrow. If the buffer is read only, (RO) is displayed.

You enter command names and responses to command prompts in the *minibuffer*. You can tell if you are in the minibuffer because the keyboard cursor moves there. The minibuffer is usually a three-line-high version of the editor. It works exactly like an editor buffer window. You can edit your input and scroll in it. When you enter the minibuffer, the right side of the mode line presents messages about the operation you are performing. The following messages may appear:

- **Completion** — If you press the ESCAPE key when you see this message, Zmacs provides help on completing command names, pathnames, and responses to commands.
- **Extended Search Characters: CTRL-H** — If you press CTRL-H HELP when you see this message, a list of extended search characters appears. You use these to perform searches.

Mouse Documentation Window

2.4.3 The mouse documentation window is in reverse video below the mode line window. Zmacs uses the mouse documentation window to provide documentation on the options available with the mouse. For example, if you are in the editor buffer window, the mouse documentation window tells you that two right clicks (R2) invoke the System menu and that one right click (R) invokes the Editor menu.

System Status Line

2.4.4 The system status line, the bottom line on the screen, provides information on the status of the entire system. In addition to the date, time and user name, two items appear there to provide status information on Zmacs:

- **Package name** — A name followed by a colon appears to the right of the user name. This tells you what package a file is using. All files use a particular package. In Figure 2-1, the package name is USER.
- **File information** — During file transfers, the file information appears to the right of the package name. This information includes the name of the file being transferred and the percentage of transfer complete. An arrow indicates input to or output from the named file.

Scroll Bar

2.4.5 The scroll bar, which is next to the left margin, allows you to scroll the information in the editor buffer window. The scrolling icons in this scroll bar region indicate whether you need to scroll forward or backward to find additional material. For example, a down arrow appears in the lower left corner if there is more information below. Similarly, an up arrow appears in the upper left corner if there is more information above. The mouse documentation window describes the scrolling operations you can perform when the mouse is positioned in the scroll bar region. For specific information about scrolling with the mouse, refer to the *Introduction to the Explorer System*.

Typeout Window

2.4.6 The typeout window returns information in response to commands or errors. It overlays the editor buffer window, starting at the top. The typeout window is a temporary window; it overwrites the editor buffer window, but as soon as you erase the typeout window, the contents of the editor buffer window are restored. Figure 2-2 shows a sample typeout window.

When you execute a command such as the List Fonts command to see the available fonts in the system, the typeout window displays the list.

The general rules for removing the typeout window are as follows:

- If the typeout window returns information because of an error, you can press the ABORT key to remove the typeout window in most cases.
- If the typeout window returns information in response to a command and the word More does *not* appear on the bottom of the window, you press the space bar or the ABORT key to remove the window. Any other key, including RUBOUT, affects your buffer as if no typeout window were present. Typing a character such as *g* removes the typeout window, but the *g* is also inserted into your file. The space bar does not insert a space.
- If the typeout window returns information in response to a command and the word More *does* appear on the bottom of the window, you press the space bar to scroll the window. To remove the window, you press RUBOUT or ABORT. Then the message Removed appears, and the screen is restored.

Figure 2-2

Typeout Window

```

HELP:
Type one of the following characters to choose an option:
C Print out documentation for the command on a given key.
D Describe a command, specified by name.
O Print documentation of editor user option variable.
R List commands whose names contain a given string.
V List all editor options whose names contain a given substring.
U Undo the last undoable command done in the current buffer.
L List the last sixty keystrokes typed inside the editor.
S List the special characters available in this buffer.
W List all characters that invoke a given command.

You may continue choosing options until you exit.

Press the space bar to exit this command.
Any other non-option key will exit and be executed.

[
  (check-type font font 'a font object')
  (LET* ((FONT-SIZE (IF (and (font-fill-pointer font) (NOT (ZEROP (FONT-FILL-
    (FONT-FILL-POINTER FONT)
    ;;ELSE
    (LENGTH (FONT-CHAR-WIDTH-TABLE FONT))))

```

Executing Commands

2.5 Zmacs allows you to execute commands several different ways. You can use any of the following methods:

- Press keystrokes. You can use keystrokes to execute many of the commands, but not all of them. Some keystrokes such as CTRL-X are prefix commands. These commands allow you to use another group of commands after you enter the prefix command.
- Click on command names in menus. You can use the mouse to select commands from the Suggestions menus, the Editor menu, and other

Zmacs menus. To *select* a command, you position the mouse cursor on the command name until it is highlighted with a box, and then you click the left mouse button.

- Press META-X and then type the command name. Commands allowed by META-X are designed to apply to your current context in the editor.

Furthermore, you can give any Zmacs command a numeric argument. The numeric argument has different effects depending on the command. You can supply positive or negative numbers as arguments.

One of the most common uses for the numeric argument is as a counter for the number of times the command should be performed. For example, supplying the Kill Word command (META-D) a numeric argument of 20 deletes 20 words after the cursor instead of 1. The following list provides examples of entering a numeric argument of 20 for a Zmacs command by using the key-stroke method, by using the menu method, and by using the META-X method:

- Keystrokes — Press CTRL-2 CTRL-0 META-D to execute the Kill Word command 20 times. You hold down any combination of the CTRL, META, SUPER, and HYPER keys while pressing numbers. Then you press the keystroke for the command.
- Menus — Press CTRL-2 CTRL-0. Then select the Kill Word command from a menu. You can find the command listed as M-D Word → in the Suggestions menus. You need to select the Move, Copy, and Delete menu listed under Other Menus. The command is under the heading Killing (for moving).
- META-X — Press CTRL-2 CTRL-0 META-X, and then type Down Indented Line.

Section 3, Command Groups, describes META-X, numeric arguments, and prefix commands in detail. It also tells you how Zmacs can help you find and complete command names.

Scrolling

2.6 Zmacs allows you to scroll the editor buffer window with the following commands:

- Next Screen (CTRL-V or CTRL-↓) — Scrolls forward a screenful of text. If you supply a numeric argument, CTRL-V or CTRL-↓ scrolls forward the specified number of *lines*.
- Previous Screen (META-V or CTRL-↑) — Scrolls backward a screenful of text. If you supply a numeric argument, META-V or CTRL-↑ scrolls backward the specified number of *lines*.

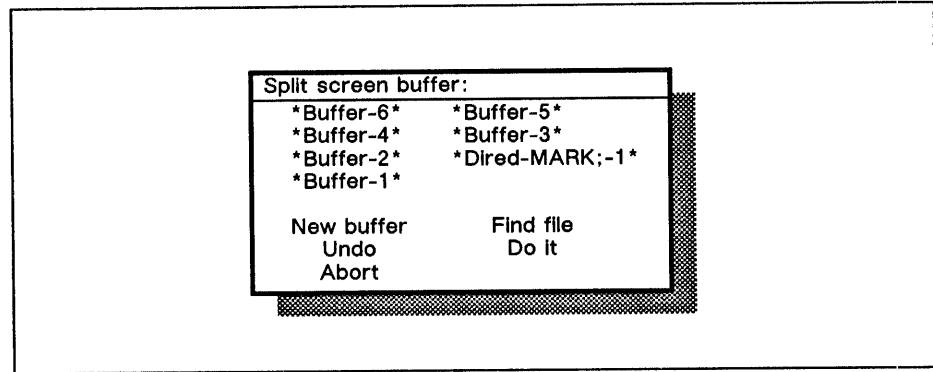
Splitting the Screen

2.7 You can divide the editor buffer window into two or more windows and edit a different buffer in each window. However, you can perform an operation in only one window at a time.

To split the editor buffer window, you execute the Split Screen command (META-X Split Screen). (You can also find this command on the Editor menu.) Figure 2-3 shows the display that appears when you execute the Split Screen command.

Figure 2-3

Split Screen Display (First Screen)



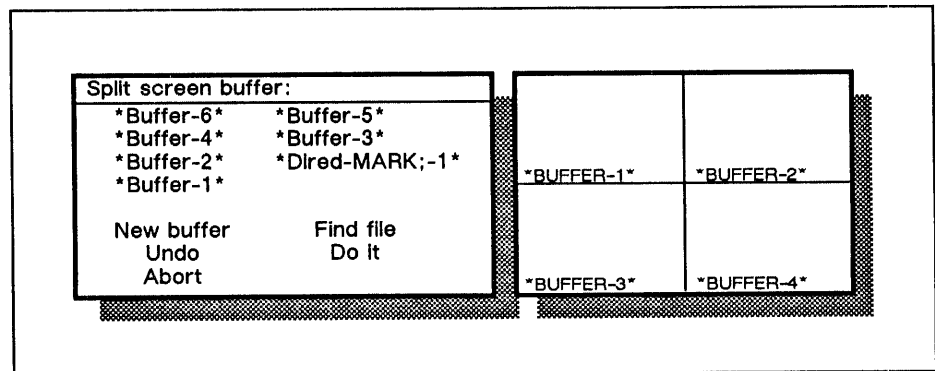
You can choose one of the buffers listed by clicking on its name, select `new buffer` to create a new buffer, or select `find file` to bring a file into a buffer for editing. When you specify the buffer or file, an empty box appears beside the Split Screen display with the buffer name on the bottom. When you specify the next buffer you want, the box is split in half. This box is divided among the buffers until you click on one of the following:

- `Do It` — Executes the command.
- `Undo` — Allows you to undo the buffers displayed. It removes one buffer name at a time from the box, starting with the last buffer you chose.
- `Abort` — Aborts the Split Screen command.

Suppose you want to split the screen among four buffers. Figure 2-4 shows an example Split Screen display after four buffers have been chosen.

Figure 2-4

Split Screen Display (Second Screen)



When you finish executing the Split Screen command, you can perform the following operations:

- Move to another window by using one of these two methods to move to another window:
 - Execute the Other Window command (CTRL-X O). If several windows are on the screen, this command moves through them all in cyclical order. Specify the window to go to by using a numeric argument, counting from 1 at the top.
 - Move the mouse cursor to another window and click any button.
- Scroll another window by using the Execute the Scroll Other Window command (META-CTRL-V). This command scrolls the other window forward one screen of text. If several windows are on the screen, this command finds the other window to scroll by going from left to right or by going back to the top left window if you are in the lower right window. Specify the number of lines to scroll forward or backward by using a positive or negative numeric argument, respectively.
- Return to displaying only one window by using the Execute the One Window command (CTRL-X 1). The window displayed is the one that the blinking keyboard cursor is in.

Menus

2.8 Zmacs contains many different types of menus that provide a wide range of operations. You can execute commands from menus by selecting them with the mouse, and many commands prompt you for information with a menu. You can also easily invoke the System menu from Zmacs. The following list summarizes the menu topics discussed:

- Types of menus — Zmacs contains several types of menus that require different methods for entering information.
- System menu — You can enter and exit Zmacs from the System menu, and you can also select options that affect Zmacs, such as Suggestions, Kill, and Reset.
- Top-Level Editor menu — This menu contains a small group of commands that allow you to perform a diverse range of Zmacs operations.
- Suggestions menus — These menus allow you to easily find and execute Zmacs commands by displaying them in functional groups.

Zmacs provides you with online help when you are using these menus. The mouse documentation window, which is displayed in reverse video, usually tells you what options you have with the mouse. If you are in the editor buffer window, the mouse documentation window tells you that two right clicks (R2) invoke the System menu and one right click (R) invokes the Editor menu. Also, if you position the mouse on one of the menu items until it is highlighted with a box, the mouse documentation window provides brief documentation on that item.

Types of Menu 2.8.1 The menus that you can invoke from Zmacs require different methods for entering information. The three basic structures of these menus are as follows:

- Standard menu
- Multiple-choice menus
- Choose-variable-values menu

Standard menus list items that you can select with the mouse. An example of a standard menu is the System menu.

Multiple-choice menus provide you with options that you select by putting an x in one of the boxes. The Kill or Save Buffers command displays a menu of this type.

Choose-variable-values menus display variables on the left and list the possible values for each variable on the right. You can choose one of the values with the mouse. The current value is displayed or highlighted. The Change File Properties command (META-X Change File Properties) displays a menu of this type (after you specify a filename).

When you move the mouse cursor off many of the menus, such as the System menu, they disappear. Other menus, such as the multiple-choice menus, display the prompts Do It and Abort, and you must click on one of these prompts to remove the menu. Some menus display the prompt Exit, and you must click on Exit to exit (for example, the Edit Screen menu on the System menu).

System Menu 2.8.2 The primary menu for the entire system is called the System menu. You can invoke the System menu no matter where you are in the editor by double clicking right with the mouse. The System menu contains a list of several other menus and programs. While most of these menus do not directly affect the editor, some of them allow you to perform operations with the editor. The following options affect the editor:

- Zmacs Editor, Create, and Select — These options allow you to enter Zmacs. Refer to paragraph 2.2, Entering Zmacs, for more information.
- Programs — You can exit Zmacs by selecting a program such as the Lisp Listener.
- Edit Screen — This option allows you to reshape the editor window. Refer to the *Introduction to the Explorer System* for information on how to use this option.
- Edit Attributes — Selecting this option invokes a choose-variable-values menu. This menu allows you to change the attributes of whatever you have on the screen at the time. You can apply this menu to a variety of windows, including the editor. Refer to the *Introduction to the Explorer System* for information on how to use this menu.

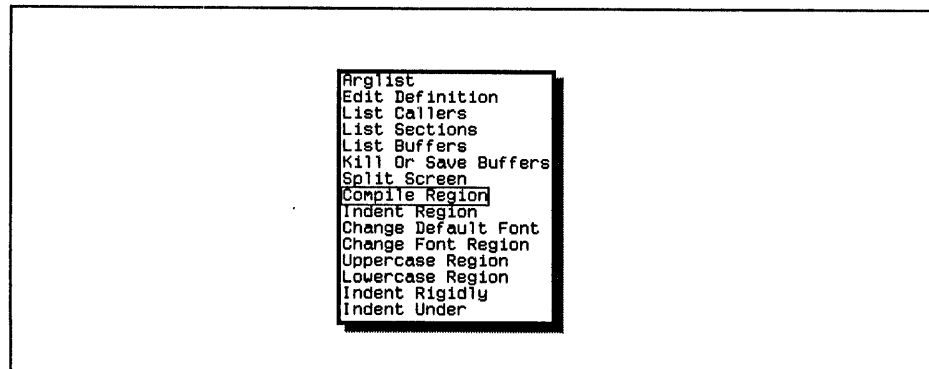
To invoke one of the items on the System menu, you position the mouse cursor on that item and click left.

Top-Level Editor Menu

2.8.3 The Top-Level Editor menu provides a small group of commands that allow you to perform such diverse operations as compiling, changing fonts, indenting code, listing and saving buffers, and editing the definition of a function. When you are in the editor, you can invoke the Editor menu by clicking right. Figure 2-5 shows the Editor menu. Most commands on the Editor menu are also available on a key. For detailed information on these commands, refer to Section 3, Command Groups.

Figure 2-5

Editor Menu



Zmacs Suggestions Menus

2.8.4 The Zmacs Suggestions menus, located to the right of the editor buffer window, make it easy for you to find and execute a command. These menus display commands and other menus that are functionally related. When you move the mouse cursor over a menu or command name, the mouse documentation window provides a brief explanation of that menu or command. You show the menu or execute one of the commands by clicking left.

Executing simple commands such as Forward Character takes longer when you use the Suggestions menus than when you use keystrokes. For the Forward Character command, it is faster to use the keystroke CTRL-F. However, the Suggestions menus are intended to be a learning aid. The functional grouping of commands makes it easy to learn what commands are available.

The Suggestions menus provide an easy method to execute commands that have long names and that do not have associated keystrokes (for example, the Create Directory command). Otherwise, you must press META-X and then type the name of the command. (Zmacs can help you complete the typing of the name by using a process called *completion*. You type enough characters to uniquely identify the command and then press ESCAPE.)

To turn on the Suggestions menus, you select the item Suggestions Menus On in the command line of the Lisp Listener or select the item Suggestions from the User Aids column of the System menu.

Help Facilities

2.9 Zmacs provides help for almost every situation. When you press the HELP key, you receive help on whatever operation you are performing. The following list summarizes the help facilities discussed in subsequent paragraphs:

- **Commands on the Zmacs Help menu** — These commands provide a wide variety of options from documentation on keys and commands to undoing a previous command.
- **Help commands for Lisp code** — These commands provide you with useful information about Lisp code, such as the definition of a function and the list of all functions that contain a specified string.
- **Aborting** — Zmacs provides commands to abort most operations.
- **Troubleshooting** — If the editor malfunctions, you can try several procedures to reactivate it.

**Help Menu
Commands**

2.9.1 If you press HELP while you are editing, a menu containing a list of help commands appears. The following table provides a brief description of the commands. For detailed descriptions of the commands, refer to the Help command group in Section 3, Command Groups.

Keystroke	Command Name	Description
HELP C	Document Command	Displays documentation for the command on a given key.
HELP D	Self Document	Displays documentation for a command.
HELP O	Describe Variable	Displays documentation for a Zmacs user variable.
HELP A	Apropos	Lists commands whose names contain a given string. The display is mouse-sensitive. You can execute a command by selecting it with the mouse.
HELP V	Variable Apropos	Lists Zmacs user variables whose names contain a given string. The display is mouse-sensitive. When you click left on any variable listed, its value appears for editing in the minibuffer. You can also click right on any variable to invoke a menu that provides documentation on the variable or allows you to modify the variable.
HELP L	List Keystroke History	Lists the last 60 keystrokes you typed.
HELP U	Undo	Undoes the last command. (An alternative keystroke for HELP U is the UNDO key.)
HELP W	Where Is	Tells what keystroke is associated with a command.
HELP S	Symbol Help	Lists the keystrokes for characters that are not on keycaps, such as \geq .
Space bar	—	Removes the help screen.

Help Commands for Lisp Code

2.9.2 Several Zmacs commands provide information about Lisp code that you may find helpful, such as the definition of a function and the list of all functions that contain a specified string. The following table briefly describes these commands. Section 3, Command Groups, discusses these commands in more detail.

Keystroke	Command Name	Description
META-X Arglist	Arglist	Lists the arguments for the function that you specify. The list is displayed in the minibuffer.
CTRL-SHIFT-V	Describe Variable at Point	Displays information about the variable at or before the cursor.
META-.	Edit Definition	Edits the definition of a function that you specify.
META-ESCAPE	Evaluate Minibuffer	Evaluates a form from the minibuffer.
META-SHIFT-E	Evaluate Region Verbose	Evaluates the current region or definition.
META-X Function Apropos	Function Apropos	Lists functions whose names contain the string that you specify.
META-X List Callers	List Callers	Lists functions that use the specified function.
META-X List Combined Methods	List Combined Methods	Lists all methods used for a specified operation on a specified flavor.
META-X List Flavor Methods	List Flavor Methods	Lists all the methods of a flavor.
META-X List Methods	List Methods	Lists all flavors with methods for a specified message.
META-SHIFT-D	Long Documentation	Provides long documentation for the function that you specify. The documentation includes the function's arguments.
CTRL-SHIFT-A	Quick Arglist	Lists the arguments for the function to the left of the cursor. The list is printed in the minibuffer.
CTRL-SHIFT-D	Quick Documentation	Displays documentation for the function being called from where the location of point.

The `describe` function provides information about any object x (except for array contents). This function recognizes arrays, symbols, floating-point numbers, packages, stack groups, closures, and FEFs, and prints the attributes of each in human-readable form.

For more information on the `describe` function, refer to *Explorer Tools and Utilities*. For more information on the terms introduced here, such as floating-point numbers, stack groups, closures, and FEFs, refer to the *Explorer Lisp Reference*.

Aborting 2.9.3 Zmacs contains commands to abort most operations. The following commands are available:

- CTRL-G — Allows you to correct editing errors, unmark a region, and clear partially entered commands (that are not executing yet) from the minibuffer.
- ABORT — Aborts commands that are waiting for input.
- CTRL-ABORT — Aborts almost any operation, such as evaluations, compilations, and file transfers.
- META-CTRL-ABORT — Aborts what CTRL-ABORT does and sometimes more. It is the most powerful Abort command and should be used only as a last resort.

Troubleshooting 2.9.4 If the editor malfunctions, you can try the following procedures to reactivate it:

- Position the mouse in the editor buffer window and click left. You should try this procedure first if, while using the editor, you press keys and nothing happens. This action may reactivate the editor.
- Press CTRL-ABORT or META-CTRL-ABORT. Pressing CTRL-ABORT or META-CTRL-ABORT allows you to abort editor conditions that hang the editor, such as evaluating a region that contains an infinite loop.
- Press SYSTEM CTRL-E or invoke the System menu and start a new editor. Pressing SYSTEM CTRL-E creates a new editor. You can also create a new editor by selecting Zmacs Editor on the System menu. These two methods are basically the same, except that if the keyboard is frozen, you can use the System menu method with the mouse. If the mouse is frozen, you can use the keyboard method. These methods let you save all your buffers with changes back to disk. You may be able to continue editing, but it is safer to save your files first.
- Use Abort in the error handler, which also provides troubleshooting information. The error handler is not directly part of the editor, but the editor does use it. For information on the error handler, refer to *Explorer Programming Tools and Utilities*.

- Enter the form `(zwei:save-all-files)` from a Lisp Listener. The function `zwei:save-all-files` is useful in emergencies when you have modified material in buffers that need to be saved but the Zmacs editor is malfunctioning. This function does in Zmacs what the Save All Files command does, but it works even when the window system is broken.
- Enter the form `(ed 'zwei:reload)` from a Lisp Listener. This form reinitializes the editor (that is, all instances of the editor). Because it deletes all existing buffers, you should use this form only as a last resort.

Files, Buffers, and Directories

2.10 Zmacs makes a distinction between files and buffers. *Files* store information on disk. Zmacs does not edit files. When you want to edit a file, Zmacs copies the file on disk into a *buffer*. Then you work with the copy in the buffer.

You can have several different buffers, each containing text from different files. The editor can switch between buffers.

When you create an editor, it automatically includes one empty buffer. This buffer usually has the same number as the editor. That is, editor 1 has a buffer called `*BUFFER-1*`. (Remember you can create more than one editor, although you usually create more buffers.)

When you specify a file to edit, Zmacs copies the file into a buffer, and by default, the buffer name is the same as the filename. The host and directory name are ignored. You can specify a different buffer name by using the Rename Buffer command (`META-X Rename Buffer`).

All files are stored in directories, which form a tree structure. A *directory* is a file that contains other files or subdirectories. A *subdirectory* is a directory within another directory. A subdirectory can also contain files and subdirectories. Zmacs provides a directory editor, *Dired*, that allows you to perform a wide range of operations on a directory.

The following list describes the topics discussed in subsequent paragraphs about the file and buffer system:

- Pathname structure — Provides a brief description of the components of an Explorer pathname.
- File types — Describes the various file types that Zmacs recognizes.
- Mode line information — Discusses some useful status information about the buffer.
- Lisp code in buffers — Describes how a Zmacs buffer containing Lisp code is divided into sections, each containing a definition, so that you can locate definitions quickly and perform operations on only the changed definitions.
- Creating directories — Tells how to create a directory.
- Creating buffers — Tells how to create a buffer.
- Listing and editing buffers — Describes how to list and edit buffers.

- Finding a file for editing — Describes how to bring a file into a buffer for editing.
- Saving and writing files — Tells how to write buffers to disk (that is, to a file stored on disk).
- Editing a directory — Tells how to invoke the directory editor (Dired) and perform a wide range of operations on the directory, such as bringing files into buffers for editing, deleting files, viewing files, and comparing files.

Pathname Structure

2.10.1 Zmacs uses the same file system (FS) that the other utilities on the system use. The *Explorer Input/Output Reference* discusses pathnames in detail. The following table shows the syntax for an Explorer pathname and provides a brief review of the components of an Explorer pathname.

Pathname syntax: *host:dir.sdir.smdir; name.type#version*

Pathname Component	Description
<i>host:</i>	Host name, followed by a colon. The top level of the pathname structure is the host name, which is usually the name of some machine.
<i>dir.sdir.smdir;</i>	Directory (<i>dir</i>) and, optionally, subdirectory names (<i>sdir</i> and <i>ssdir</i>), followed by a semicolon. Under the host name are directories, which can contain subdirectories and files. A subdirectory can also contain subdirectories and files. The bottom level of the structure contains either a file or a directory without any files.
<i>name</i>	Filename.
<i>.type</i>	File type preceded by a period. The editor looks at the file type and activates an appropriate mode for that file type. If the file type is text, the editor puts you in Text mode. Refer to paragraph 2.10.2, File Types, for more information.
<i>#version</i>	The number sign (#) signifies that the version number follows. Each time you change a file and write it to disk, you do not overwrite the old file. You simply make a new copy and store it with a higher version number (one higher) than the file you brought in. You can specify the version number for the new copy if you want. However, you cannot specify an existing version number. A greater than sign (>) following the number sign (#) indicates the greatest version or the most current.

The Explorer system also understands other types of pathnames and determines the syntax from the type of machine. Pathnames for other machines may contain a pathname component named *device*, followed by a semicolon, after the host name. The Explorer file system does not currently use the

device component but recognizes it in a pathname for other types of machines.

Zmacs contains some elaborate defaulting conventions for pathnames:

- The mode line tells you the default pathname. To accept the default, you press RETURN.
- When you type only part of the pathname and press ESCAPE, Zmacs, using completion, shows the rest of the pathname as far as it is unique. When you are satisfied with the pathname, press RETURN.
- When you type only part of the pathname and press END, Zmacs completes the pathname and exits if it is unique.
- When you want to change parts of a pathname, you do not always need to type the entire pathname. Suppose you have the following pathname:

EX1: ZWEI; MENUS.LISP#22

If you want to change only the filename MENUS, you simply type a new filename, such as CATS. When you change any other component of the pathname, you need to supply the adjacent punctuation. You should always place the punctuation on the side toward the filename. The following list shows an example of how to change each component of the pathname in the example:

Pathname Component	How to Change
EX1	EX2:
ZWEI	WINDOW;
MENUS.	CATS.
LISP	.XLD
#22	#8

If you want to change two adjacent components, such as the host and directory names, you can type them together (for example, EX2: WINDOW;). However, if the components are not adjacent, you need to type the entire pathname.

When executing one of the file commands, you can also yank the default pathname into the minibuffer and edit it. You can use the cursor movement commands and the delete commands in the minibuffer. To yank the default pathname, you press CTRL-SHIFT-Y.

File Types 2.10.2 Zmacs makes distinctions between files of different types. It provides different capabilities depending on the file type. Zmacs looks at the file type in the pathname and activates an appropriate mode for that file type. For example, if the file type is Lisp, Zmacs puts you in Common Lisp mode when you edit the file. (However, if the attribute list in the top line of the file lists a different mode, such as Zetalisp, the editor uses that mode.)

The file type is also a label so that you can tell what kind of file it is. You can have as many types of files as you want. The following file types are common ones:

- Lisp — Contains Lisp code. When you want to edit a Lisp file, Zmacs sees that the file type in the pathname is Lisp and automatically activates Common Lisp mode (unless the attribute list overrides it with a different mode, such as Zetalisp).
- Text — Contains text. When you want to edit a text file, Zmacs sees that the file type in the pathname is text and automatically activates Text mode.
- Xld — Contains compiled code.
- Temporary — Is automatically deleted from a directory if you clean up the directory with the Clean Directory command or the Dired Automatic command. Temporary files are deleted regardless of how many copies of the files you have. If you have only one copy of the file, it is deleted. You can find the list of temporary files in the Zmacs user variable Temp File Type List.
- User-defined file type — Is a file type you create. For example, you can have a file type called cats. This name does not mean anything to the system, but it does not mind what file type you specify. Zmacs uses a default mode called *Fundamental mode* for user-defined file types.

Mode Line Information 2.10.3 The mode line provides the following useful information about the buffer:

- That you are in Zmacs.
- Which mode you are in. Common Lisp mode is the default mode. (Paragraph 2.12, Setting Modes and Buffer Attributes, describes modes.)
- The name of the buffer you are editing. If the buffer is connected to a file (that is, the buffer was read from disk), the version number of the file appears in parentheses beside the pathname.
- If more of the buffer is above or below what you see on the screen, an up arrow or down arrow appears on the left side of the mode line. When the top of the buffer is on the screen, the up arrow disappears because there is no more of the buffer above; however, the down arrow stays. When the bottom of the buffer is on the screen, the down arrow disappears and the up arrow stays.
- If the buffer has been changed since it was read from disk, an asterisk appears on the left side of the mode line. If you write the buffer back to disk, the asterisk disappears.

**Lisp Code
in Buffers**

2.10.4 When Zmacs reads into a buffer a file that contains Lisp code, it divides the definitions in the buffer into sections. The only buffers that have sections are those containing Lisp code. Other buffers do not have sections. A *section* can contain any defining construct that begins with **def** after the opening parenthesis in the first column, such as **defvar**, **defflower**, and **defun**. The defining construct can be user-defined. Each definition is a section.

The section scheme allows you to locate definitions quickly and perform operations only on the changed sections (or definitions). For example, you can perform these operations:

- List all the sections (definitions) in a specified buffer.
- List and edit the definitions that have been changed in a specified buffer or in all buffers.
- Compile and evaluate only the changed definitions in a specified buffer or in all buffers.

When Zmacs sectionizes a buffer containing Lisp code, it actually looks for a matching set of parentheses starting with an opening parenthesis in the first column. Zmacs' detection of an opening parenthesis in the first column is unconditional. Even if the parenthesis is inside a quoted string or a block comment (that is, `#| ... |#`), Zmacs starts a new section. This feature may cause unexpected errors.

Note that Zmacs finds an opening parenthesis only if it follows a return character. A long line that wraps on the screen so that an embedded parenthesis appears on the left edge of the screen does not start a new section.

If a parenthesis is in the first column of a string, you can prevent Zmacs from starting a new section by putting a backslash in front of the parenthesis in Common Lisp mode or a forward slash in Zetalisp mode. The backslash (or forward slash) protects the next character in the string and is effectively removed when the string is originally read.

Code that is commented out is ignored for making sections, unless the code is commented with the **comment** function.

At the top of the buffer is a section called the *buffer header*. Essentially, the buffer header contains the attribute list. Zmacs names the other types of sections as follows:

- If the section is a definition, Zmacs names the section according to the name of the definition. For example, if the definition is a **defun** defining **com-forward-character**, Zmacs names the section **com-forward-character**.

- If the section begins with an opening parenthesis but it is not a definition, Zmacs names the section with the following syntax:

name-form-number

where:

name is the name of the buffer.

form is the first word of the form.

number is a number that Zmacs assigns to the section.

For example, `*BUFFER-1*-setq-2` represents a buffer named `*BUFFER-1*`, a form starting with `setq`, and a section with a number of 2.

Creating Directories

2.10.5 If you want to create a directory, you can use the Create Directory command. You press `META-X` and then type `create Directory`; a prompt appears in the minibuffer asking you for the pathname of the directory. You need to type the pathname only through the semicolon (`;`).

Creating Buffers

2.10.6 Zmacs automatically creates a buffer for you when you enter the editor. This buffer usually has the same number as the editor. That is, editor 1 has a buffer called `*BUFFER-1*`.

You can create a new buffer by using the Select Buffer command (`CTRL-X B`). You specify a buffer name that does not already exist and press `RETURN`. (For more information on this command, refer to the Buffer commands in Section 3, Command Groups.)

You can create as many buffers as you want.

NOTE: When you update a buffer in one editor and look at it in another editor, it is updated in the second editor also. Only one copy of that buffer exists. You do not have two different copies with one out of date in relation to the other.

Listing and Editing Buffers

2.10.7 Zmacs provides many commands that allow you to list and edit buffers. You can use the following commands:

- List Buffers (`CTRL-X CTRL-B`) — Lists the buffers you have and allows you to choose one for editing. You can also perform other operations on the buffer, such as compiling.
- Select Previous Buffer (`META-CTRL-L`) — Selects the previous buffer for editing.
- Select Buffer (`CTRL-X B`) — Allows you to select the buffer of your choice for editing.

The List Buffers command (CTRL-X CTRL-B) shows the Zmacs buffer names. It lists all the buffers you have. You can then select a buffer for editing. You position the mouse cursor on the buffer you want to edit and then click left. If you click right on the buffer name, a menu appears showing other operations that you can perform on the buffer:

- **Compile File** — Compiles the file to which the buffer is connected on disk and puts the compiled version on disk with the same name and a different type, xld. It does not change the contents of memory. If you have made changes to the buffer, first you are asked if you want to save the buffer so that your changes will be in the file on disk.
- **Kill** — Deletes the buffer. If the buffer is connected to a file and you have modified the buffer, you are asked if you want to save the buffer first. If you do not save it, the original version on disk is unchanged.
- **Print** — Prints the buffer on a line printer.
- **Unmod** — Marks the buffer as unmodified. The changes you made to the buffer are still in the buffer, but Zmacs does not treat the buffer as modified. The effect is that Kill or Save Buffers will not mark the file to save.

You may want to do this if you are using Kill or Save Buffers several times and you are not ready to save that particular buffer. Unmod prevents you from having to remove the x from the Save box each time you use the command. When you are ready to save the buffer, you can then mark the Save box.

- **Save** — Writes the buffer to disk, using the pathname of the file from which the buffer originally came and incrementing the version number. The Save operation automatically gives you the default filename. You do not specify the filename.
- **Write** — Writes the buffer to the filename you specify.
- **Select** — Allows you to edit the buffer.

You can also execute the List Buffers command from the Editor menu or the Suggestions menus. For detailed information on the List Buffers command, refer to the Buffer commands in Section 3, Command Groups.

The Select Previous Buffer command (META-CTRL-L) allows you to select the previous buffer for editing. If you want to go back more than one buffer, you enter a numeric argument with the command. For example, if you want to go back three buffers, you press META-3 META-CTRL-L. The current buffer is 1.

The Select Buffer (CTRL-X B) command allows you to select a buffer of your choice for editing. When it asks you which buffer you want to edit, you enter a name. If you enter the name of a buffer that does not exist, it tells you it does not exist. If you press RETURN again, it creates a buffer with the name you specified.

**Finding a File
for Editing**

2.10.8 To bring a file into one of your buffers for editing, you enter the Find File command (CTRL-X CTRL-F). You are prompted in the mini-buffer for the name of the file you want to edit.

The Edit Definition command (META-.) also brings in a file for you to edit. When you specify a function name, the command brings in the file(s) with the definition.

**Saving and
Writing Files**

2.10.9 The Save File command (CTRL-X CTRL-S) and the Write File command (CTRL-X CTRL-W) allow you to write a buffer to disk. The Save File command writes the buffer to disk, using the pathname of the file from which the buffer originally came and incrementing the version number. This command automatically provides the default file pathname. Thus, you do not need to specify the filename. If the buffer is new or you want to specify a different filename, you must use the Write File command. If you accept the default, this command also works like the Save File command.

You can also save a buffer by using the List Buffers menu or the Kill or Save Buffers menu. You can find both of these menus on the Editor menu and on the Suggestions menus.

NOTE: When you exit the editor, your files are not automatically saved to disk. You must save them with one of these commands. Your edit sessions are not lost unless you execute the Revert Buffer command (CTRL-X CTRL-R), cold boot, or perform a **disk-restore**. (You can even log out and your edit sessions stay.) Thus, you do not need to save buffers on disk each time you want to exit the editor. (*Explorer Input/Output Reference* describes **disk-restore**.)

Editing a Directory

2.10.10 Zmacs provides a directory editor (Dired) that allows you to perform a wide range of operations on a directory. Dired displays a listing of all the files and subdirectories in a directory. To enter Dired, you press CTRL-X CTRL-D and enter the name of a directory. If you press HELP M, a brief description of the Dired commands appears. You enter these commands, which are single letters, beside the name of the file for which you want to perform an operation.

To position the keyboard cursor beside a filename, you can use the following methods:

- Press CTRL-P or ↑ to move the cursor up a line.
- Press CTRL-N, ↓, or the space bar to move the cursor down a line.
- Position the mouse cursor where you want the keyboard cursor and click left.

The following table summarizes some of the important Dired commands. Section 3, Command Groups, explains Dired in detail.

Command	Description
E	If the keyboard cursor is on a line that is a file, edits the file. If the keyboard cursor is on a line that is a directory name, starts a new Dired on that directory.
V	Views the file.
D	Marks the file for deletion.
U	Removes the delete mark.
B	Compiles the file.
R	Renames the file.
P	Prints the file.
S	Lists the files in the subdirectory.
Q	Performs operations such as deletions on the files you have marked and then exits Dired.
X	Performs operations such as deletions on the files you have marked and leaves you in Dired.
=	Compares the file with the highest version number of the file.
ABORT	Aborts Dired.

Cursor Movement

2.11 Zmacs contains many cursor movement commands. Depending on whether you are in Text, Common Lisp, or Zetalisp mode, these commands allow you to move by text quantities such as word and sentence or by Lisp quantities such as symbolic expression and list. The commands specific to Common Lisp mode work the same in Zetalisp mode. Many commands, such as moving up or down a line, apply to all three modes. The following list summarizes the cursor movement topics discussed in subsequent paragraphs:

- General rules for cursor movement — These rules illustrate some patterns for learning the keys that move the cursor.
- Cursor movement by quantity — The cursor movement commands are divided into groups according to the quantity by which they move the cursor. A *quantity* is a character, word, symbolic expression, list, paragraph, and so on.
- Storing cursor locations — Zmacs allows you to store cursor locations and return to them later.

NOTE: You can add numeric arguments to the cursor movement commands to execute more than one operation at a time. For example, you can move three characters forward instead of one by adding a numeric argument of 3 to the Forward Character command (CTRL-F or →). You press CTRL-3 CTRL-F.

The following list defines the quantities by which the cursor movement commands move the cursor:

- *Character* — Letter, number, or special symbol, such as a punctuation character.
- *Real line* — Synonymous with the term *line*, which is ended by a return character. A line can span multiple lines on the screen before the return character appears. (Zmacs recognizes return characters, but they are not displayed in the buffer.)
- *Word* — A group of contiguous letters containing no blanks.
- *Symbolic expression* — A Lisp expression enclosed in parentheses or a single symbol, such as `my-symbol`.
- *Sentence* — Ended by a period (.), question mark (?), or exclamation point (!) followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote ("), single apostrophe ('), closing parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.
- *Definition* — Any defining construct that begins with an opening parenthesis in the first column followed by `def`, such as `deffavor`, `defmethod`, and `defvar`.
- *Paragraph* — Delimited by blank lines or by lines that start with a delimiter in the Zmacs user variable Paragraph Delimiter List or in the user variable Page Delimiter List. You can see and change the values of these variables by executing the Variable Apropos command (HELP V).

General Rules for Cursor Movement

2.11.1 Some general rules exist for the keys that allow you to move the cursor. These rules help you to learn patterns for the way the keystrokes are used in the commands.

The following table shows the basic keys used for the cursor movement commands and their meaning. These keys by themselves do not move the cursor.

Key	Meaning
F	Forward
B	Backward
N	Next
P	Previous (many commands say Up instead of Previous)
A	Beginning
E	End

These keys are used in conjunction with the META and CTRL keys, as described in the three tables that follow.

If you hold down CTRL and press one of the above keys, the cursor is moved on the lowest level (that is, by the smallest quantity). The corresponding arrow keys are also listed. The following table describes the commands these keys perform:

Key	Meaning
CTRL-F or →	Forward Character
CTRL-B or ←	Backward Character
CTRL-N or ↓	Down Real Line
CTRL-P or ↑	Up Real Line
CTRL-A or SUPER-←	Beginning of Line
CTRL-E or SUPER-→	End of Line

If you hold down META and press one of these keys, the cursor is moved on the next higher level (that is, by the next higher quantity). The corresponding arrow keys (if available) are also listed. The following table describes the commands these keys perform:

Key	Meaning
META-F or META-→	Forward Word
META-B or META-←	Backward Word
META-A or META-↑	Backward Sentence
META-E or META-↓	Forward Sentence

If you hold down META-CTRL, the keys move the cursor by Lisp forms in Common Lisp or Zetalisp mode. The corresponding arrow keys (if available) are also listed. The following table describes the commands these keys perform:

Key	Meaning
META-CTRL-F or META-CTRL-→	Forward Sexp
META-CTRL-B or META-CTRL-←	Backward Sexp
META-CTRL-N	Forward List
META-CTRL-P	Backward List
META-CTRL-[or META-CTRL-↑	Beginning of Definition
META-CTRL-] or META-CTRL-↓	End of Definition

**Cursor Movement
Commands Grouped
by Quantity**

2.11.2 The cursor movement commands allow you to move by character, word, symbolic expression, line, list, definition, sentence, paragraph, buffer, and screen. The following table summarizes these operations:

Quantity	Key	Meaning
Character	CTRL-B or ← CTRL-F or →	Backward Character Forward Character
Word	META-B or META-← META-F or META-→	Backward Word Forward Word
Symbolic expression	META-CTRL-B or META-CTRL-← META-X Backward Sexp No Up META-CTRL-F or META-CTRL-→ META-X Forward Sexp No Up	Backward Sexp Backward Sexp No Up Forward Sexp Forward Sexp No Up
Line	META-M CTRL-A or SUPER-← META-N META-X Down Indented Line CTRL-N or ↓ CTRL-E or SUPER-→ META-P META-X Up Indented Line CTRL-P or ↑	Back to Indentation Beginning of Line Down Comment Line Down Indented Line Down Real Line End of Line Up Comment Line Up Indented Line Up Real Line
List	META-X Backward Down List META-CTRL-P META-CTRL-(META-CTRL-D META-CTRL-N META-CTRL-) META-)	Backward Down List Backward List Backward Up List Down List Forward List Forward Up List Move Over)
Definition	META-CTRL-[or META-CTRL-↑ META-CTRL-] or META-CTRL-↓	Beginning of Definition End of Definition
Sentence	META-A or META-↑ META-E or META-↓	Backward Sentence Forward Sentence
Paragraph	META-[META-]	Backward Paragraph Forward Paragraph
Buffer	META-< or HYPER-↑ META-> or HYPER-↓	Goto Beginning (of buffer) Goto End (of buffer)
Screen	SUPER-↓ SUPER-↑	Move to Bottom of Screen Move to Top of Screen

**Storing
Cursor Locations**

2.11.3 Zmacs allows you to store cursor locations (or point) and return to them later. The cursor locations can be in different buffers, allowing you to jump between buffers. You store the cursor locations on the point PDL or in a register.

The *point PDL* is a push-down list that stores cursor locations. New entries are added to the top of the point PDL. When the point PDL is full, entries on the bottom are deleted as new entries are added.

Registers are storage locations in which you can store a cursor location and/or a block of text. You can store only one cursor location at a time in a register. However, the cursor location stays in a fixed place, whereas the cursor locations on the point PDL are dynamic. (The cursor locations are moved down the point PDL as new entries are added.)

Point is the position in the file represented by the left edge of the keyboard cursor. Point is never on a character like the keyboard cursor is; it is always between characters. For example, if the cursor is on the *e* in the word *me*, point is between the *m* and the *e*.

The following list summarizes the topics discussed in subsequent paragraphs about storing cursor locations:

- Storing cursor locations on the point PDL — Describes how to store, view, and return to previous cursor locations.
- Storing cursor locations in a register — Describes how to store, view, and return to a cursor location stored in a register.

*Storing
Cursor Locations
on the Point PDL*

2.11.3.1 You can store, view, and retrieve cursor locations stored on the point PDL. In addition to the commands that you use to store cursor locations, Zmacs contains many commands that automatically put points on the point PDL. You can tell when a command puts a point on the point PDL because the mode line window displays the message Point Pushed whenever a cursor location is saved.

The default of the point PDL is 8 locations. If you want to change this default, refer to Section 4, Customizing Zmacs. You can change this number to any value by changing the Zmacs user variable Point PDL Max with either the Set Variable command (META-X Set Variable) or the Variable Apropos command (HELP V).

The point PDL lists three lines of each cursor location saved. You can list more lines when you execute the Show Point PDL command by giving it a numeric argument.

The following list summarizes the operations you can perform with the point PDL:

- Put the current cursor location on the point PDL — You use the Push Pop Point Explicit command (META-space bar).
- Move to the previous location in the point PDL — You use the Move to Previous Point command (META-CTRL-space bar). This command exchanges the current location with the previous one.
- Move to the location before the previous location on the point PDL — You use the Move to Default Previous Point command (CTRL-X META-CTRL-space bar). Pressing CTRL-X META-CTRL-space bar again returns to the previous entry on the point PDL. Pressing a third time returns to the original point.

- View the point PDL and return to one of the locations — You use the Show Point PDL command (META-STATUS) to view the point PDL. Then you use the mouse to select the entry that contains the location to which you want to return. You can return to any line shown. Figure 2-6 shows an example point PDL listing.

The display is mouse-sensitive except for the first line. You click left on the line to which you want to return. The first line tells you the buffer from which the cursor location came. If the buffer contains Lisp code, the first line also tells the section of the buffer from which point came. For example, it might indicate that the three lines are from a function called `com-find-file` in the buffer called `*BUFFER3*`. This information is helpful when you have a few lines of code from the middle of a function and you want to know from what function the lines came.

Figure 2-6 Show Point PDL Command

```
This is a list of locations to which you may jump.
META-STATUS gives this display; arg. controls the number of lines shown in one entry.

Other commands that manipulate the point pdl are:
META-SPACE with no numeric argument adds your current position to the list.
  With a numeric argument n, exchange the current and nth points
CTRL-META-SPACE rotates through the current & last 2 locations. Args. change 2.
The MOUSE is like CTRL-META-SPACE with an argument but rotates the other direction.
1 from Buffer header in GETTYBURG-INSERT.TEXT#> TI-EXAMPLE.ZMACS; L10:
  freedom and that government of the people, by the people, for the people,
  shall not perish from the earth.
2 from Buffer header in GETTYBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; L10:
  we have come to
  dedicate a portion of that field as a final resting place for those who here
  gave their lives that the nation might live. The world will little note,
3 from COM-FIND-FILE in ZMACS.LISP#> ZWEI; L10:
  (DEFCOM COM-FIND-FILE "Visits a file in its own buffer.
  Reads in a filename from the minibuffer. If the file is already
  in a buffer, selects that buffer. Otherwise creates a buffer
4 from Buffer header in GETTYSBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; L10:
  We have come to
  dedicate a portion of that field as a final resting place for those who here
  gave their lives that the nation might live. The world will little note,
```

Example of Saving Cursor Locations on the Point PDL Suppose you are editing in the middle of a large buffer. You want to go to the top of the buffer, examine it for a second, and return to where you were. First, you execute the Goto Beginning command (META-<) to take you to the top of the buffer, which you then examine. META-< pushes the current cursor location on the point PDL.

Next, you execute the Move to Previous Point command (META-CTRL-space bar) to return you to the previous location in the point PDL (which was the place in the middle of the buffer where you were originally).

If you edit some of the text at the top of the file, you may have another point pushed on the PDL. In this case, the Move to Previous Point command (META-CTRL-space bar) does not take you back to your original location. It

switches you back and forth between two locations at the top of the file. Instead, you should use the Show Point PDL command. Then you can select the location you want.

*Storing Cursor
Locations in Registers*

2.11.3.2 You can store a cursor location in a register and then later view it or return to it. You can store only one cursor location at a time in each register. However, the cursor location stays in a fixed place, whereas the cursor locations on the point PDL are dynamic. You can also store a cursor location and a block of text simultaneously in one register. (Paragraph 2.13.5, Storing Text in Registers, describes how to store text in a register.)

Registers have one-letter names. For example, you can name a register A, B, C, or even \$, as long as it is a single character. The limit on the number of registers you can create is the number of characters available to name them.

Storing a Cursor Location The following procedure describes how to store the current cursor location in a register:

1. Press CTRL-X S. (This command is the Save Position command.)
2. Type any character, such as A, to specify a register.

Viewing and Returning to Cursor Locations You can use the following procedures to return to stored cursor locations. The first procedure describes how to view and return to cursor locations stored in registers with the Save Position command (CTRL-X S).

1. Press META-X and type `show saved positions`. The display is mouse-sensitive.
2. Click on the line of text that contains the cursor location to which you want to return.

The following procedure allows you to return to a cursor location, but the contents of the register are not displayed:

1. Press CTRL-X J. (This command is the Jump to Saved Position command.)
2. Type any character, such as A, to specify the register that contains the cursor location you want.

NOTE: You can store a cursor location and a block of text simultaneously in the same register. You use the Save Position command (CTRL-X S) to store the cursor location and the Put Register command (CTRL-X X) to store the text. For both commands, you specify the same register name.

Setting Modes and Buffer Attributes

2.12 Zmacs provides several modes and buffer attributes that change how Zmacs operates on the buffer. If you are editing Lisp code, you use Common Lisp or Zetalisp mode. If you are editing text, you use Text mode. Some of the buffer attributes are the fonts, the package, and the width of a tab.

You can set the modes and change the buffer attributes. When you change a mode or buffer attribute, you can make the change permanent by changing the attribute list in the top line of the buffer. The following list summarizes the topics discussed in subsequent paragraphs about modes and buffer attributes:

- Modes — Describes the various modes Zmacs contains and tells you how to set them.
- Buffer attributes — Tells you how to change buffer attributes and describes the attribute list in the top line of the buffer.

Setting Modes

2.12.1 Zmacs provides several modes that change the way Zmacs operates. Zmacs contains both major and minor modes. The modes change how commands operate to provide different capabilities for the type of material on which you are working. If you are editing Lisp code, either Common Lisp or Zetalisp mode is appropriate. Text mode is appropriate for editing text. You can tell what mode you are using by looking at the mode line window, which lists major and minor modes.

A *major mode* determines the overall framework for operations on your file. Zmacs contains the following major modes:

- Common Lisp — Designed for editing Common Lisp code
- Zetalisp — Designed for editing Zetalisp code
- Text — Designed for editing text
- Fundamental — Designed for dealing with user-defined file types
- Macsyma — Designed for editing Macsyma code
- Ztop — Designed for editing at the top level

You always have one and only one major mode active for a buffer. If you are editing several buffers, you can have a different major mode for each buffer.

If you do not specify a file type in the pathname when you write a buffer to disk, Zmacs uses the canonical type corresponding to the mode of the buffer.

When you first enter the editor and have a new buffer, the default mode is Common Lisp. When you read files from disk, Zmacs looks at the file type in the pathname and activates the appropriate mode for that file type. If the file has a recognized type, such as Lisp, the default mode is the mode appropriate for that type (in this case, Common Lisp mode). The default mode for unrecognized file types is Fundamental mode.

However, if the attribute list in the top line of a file lists another mode, that mode overrides the file type. For example, if the file type is Lisp and the attribute list contains the mode Zetalisp, Zmacs uses the Zetalisp mode.

If you change the major mode with one of the mode commands, the change lasts only for that edit session unless you execute the Update Attribute List command (META-X Update Attribute List) to make the change permanent. Then, when you save the buffer to disk, the change is recorded in the attribute list. When you copy the file from disk into a buffer again, it is in the new major mode.

A *minor mode* is used in conjunction with a major mode. A minor mode also changes how Zmacs operates but on a smaller scale than a major mode. You can combine any minor mode with any major mode. You can also have more than one minor mode active at a time. Zmacs does not record minor modes in the attribute list in the top line of the file. The minor modes are temporary modes that remain active only during a particular edit session.

A minor mode can be either sticky or unsticky. A *sticky* minor mode remains active when you change major mode or go to another buffer. An *unsticky* minor mode does not remain active. Also, if you reparse the attribute list (to record a new major mode setting), the sticky minor modes remain active and the unsticky ones do not.

Zmacs contains the following minor modes:

- Any Bracket — Zmacs treats the characters [{}] like parentheses.
- Atom Word — All word commands act on Lisp atoms.
- Auto Fill — Zmacs automatically fills text.
- Electric Font Lock — Zmacs puts text in font A, comments in font B, quoted strings (commonly documentation strings) in font C, other strings in font D, and function specifications in font E.
- Electric Shift Lock — Zmacs uppercases everything except comments and strings.
- Overwrite — Typing over existing characters replaces them.
- RETURN Indents — Pressing RETURN indents the next line and LINE FEED does not.
- Uppercase Global Functions — Function names that are contained in the GLOBAL package are automatically uppercased.
- Word Abbreviation — Zmacs expands word abbreviations.

The unsticky minor modes are Electric Shift Lock mode, Electric Font Lock mode, and RETURN Indents mode.

The following list describes the procedures for setting major and minor modes:

- To change major modes or set any of the minor modes, you press META-X and then type the command name for the mode. The command name is the name of the mode and the word *mode*. For example, if you want to enter Common Lisp mode, you press META-X and type `Common Lisp Mode`. To specify Electric Font Lock mode, you press META-X and type `Electric Font Lock Mode`. Refer to the Mode and Buffer Attribute commands in Section 3, Command Groups, for a list of the command names.

If you do not type the entire command name, you can press ESCAPE and Zmacs completes the command name if it can. If you press HELP, Zmacs describes what you are doing and what the possible completions for the command name are.

- To turn off one of the minor modes, you press META-X and then type the command name. For example, to toggle the Electric Shift Lock mode on and off, you press META-X and type `Electric Shift Lock Mode`.

The mode line tells you what major and minor modes are in effect.

Setting Buffer Attributes

2.12.2 Zmacs allows you to change many attributes that affect the buffer. When you change one of the attributes, you can make the change permanent by recording it in the *attribute list* in the top line of the buffer. When you save the buffer to disk, the attribute list is also saved. When you bring the file back into a buffer, Zmacs uses the attributes recorded in the attribute list. You can choose not to record the changes in the attribute list. Then, those changes apply only for that edit session.

You can change the following attributes for the buffer:

- Major mode — Tells Zmacs which major mode to use for the buffer. To change major mode, you use one of the major mode commands, such as the Common Lisp Mode command (META-X Common Lisp Mode), the Zetalisp Mode command (META-X Zetalisp Mode), or the Text Mode command (META-X Text Mode). A prompt asks if you wish the attribute line (-*-) updated. To make the change permanent, answer *y* (for Yes). If you answer *n* (for No) to this prompt, you can go back later and execute the Update Attribute List command (META-X Update Attribute List).
- Package — Specifies the package to associate with the buffer. Each buffer has associated with it a default package. The package is used during the interning of symbols from the buffer. *Interning a symbol* means to either create the symbol or find it if there is a symbol that has the name that you typed.

The reason you change package names is to change the variables you are referencing. If a buffer makes references to variables or functions in the editor (for example), you have the choice of typing `zwei:` in front of each or simply putting the buffer in the ZWEI package and leaving the `zwei:` off all names. Another option is that you can precede each top-level form with `zwei:`.

To change packages, you use the Set Package command (META-X Set Package).

- **Fonts** — Specifies which fonts to use in the buffer. A *font* specifies the typeface and size of characters. The Explorer system contains many different fonts. You can use different fonts to highlight different parts of your buffer.

Each buffer has a set of fonts associated with it. When you create a new buffer, it automatically comes with one font: *cpfont*. This is the default font. You can then add more fonts. You can specify up to 26 different fonts for the buffer.

To set fonts, you use the Set Fonts command (META-X Set Fonts).

- **Backspace** — Specifies that backspace characters are to overprint on display. To set this attribute, you use the Set Backspace command (META-X Set Backspace).
- **Base** — Sets the base for numbers in the buffer (for example, 10 for decimal and 8 for octal). To set this attribute, you use the Set Base command (META-X Set Base).
- **Lowercase** — Specifies that the buffer contains lowercase or mixed-case data. To set this attribute, you use the Set Lowercase command (META-X Set Lowercase).
- **Nofill** — Specifies not to use Auto Fill mode without an explicit user command. To set this attribute, you use the Set Nofill command (META-X Set Nofill).
- **Patch file** — Specifies that the buffer is a patch file. To set this attribute, you use the Set Patch File command (META-X Set Patch File). This attribute allows you to redefine functions (and so on) that are already defined in other files without receiving warnings when the file is compiled or loaded.
- **Tab width** — Sets the displayed width of tab characters for the buffer. To set this attribute, you use the Set Tab Width command (META-X Set Tab Width).
- **VSP (vertical interline spacing)** — Sets the vertical interline spacing (VSP) for the buffer. The *VSP* is the number of blank rows of pixels between lines of text. To set this attribute, you use the Set VSP command (META-X Set VSP).

When you execute one of the commands to change a buffer attribute, you have the option of recording the change in the attribute list. If no attribute list exists, the command automatically creates it for you. The attribute list must be the first nonblank line in the file. (You can have blank lines before it.)

You can also edit the attribute list. The changes do not take effect until you retrieve the buffer from disk or execute the Reparse Attribute List command. Figure 2-7 shows two example attribute lists. Notice that you can specify the font syntax in two different ways.

If the attribute list does not begin with a comment character in the language you are using, you cannot compile the buffer. Zmacs ignores the comment character when reading the attribute list. It looks for the characters `--`, which define where the attribute list starts. Thus, you do not need a comment character for a text file because you do not compile text files.

The attribute list also ends with the characters `--`. In between, it has various keywords, each followed by a colon and the value of that keyword. A semicolon follows a value if another keyword follows the value. You use the following keywords when editing the mode line:

Mode	Lowercase
Package	Nofill
Fonts	Patch-file
Backspace	Tab-width
Base	VSP

If you use Common Lisp for the mode in the attribute list, you need to hyphenate it (that is, `Common-Lisp`).

In Figure 2-7, Example Attribute List, the mode is Common Lisp, the package used when internung symbols from the buffer is ZWEI (the editor package), and the fonts in the file are `medfnt` and `hl12b`.

Figure 2-7

Example Attribute List

```
;; -- Mode:Common-Lisp; Package:ZWEI; Fonts:(medfnt hl12b) --
```

Deleting and Moving Text

2.13 Zmacs provides a wide range of commands to delete, move, and copy text. Zmacs makes a distinction between deleting and killing text. When you *delete* text, you cannot recover it. The delete commands perform only small deletions, such as character deletions. When you *kill* text, you put the text you delete from the buffer on the kill history. The *kill history* is a stack of infinite size; new entries are added on the top of the stack. You can kill text as small as one character and as large as the entire buffer. You can also *save* text on the kill history without removing it from your buffer.

Usually, the text you kill is a *region* that you have marked. When you *mark* a block of text, it is highlighted (usually by underlining), and you can manipulate it as a single unit. You can mark text of any size.

After you put text on the kill history, you can *yank* (retrieve) the text back into your buffer. You can yank all or only parts of the entry. A save followed by a yank is the way Zmacs allows you to copy text. A kill followed by a yank is the way Zmacs allows you to move text.

You can also store a block of text in a *register* and yank it into your buffer at a later time. The block of text can be any size. You can store only one block of text in a register. However, it stays in a fixed place, whereas the kill history is dynamic. (That is, text is moved down the kill history as new entries are added.) You can also store a cursor location simultaneously in the same register in which you store text.

The following list summarizes the topics discussed in subsequent paragraphs about deleting and moving text:

- Exchanging text — Tells how to exchange (transpose) text, such as characters and words.
- Marking text — Tells how to mark text so that you can manipulate it as a region.
- Deleting and killing text — Tells how to remove text of any size from your buffer and put it on the kill history. Also tells how to put text on the kill history without removing it from your buffer.
- Yanking text — Tells how to copy an entry (or a line of an entry) on the kill history into your buffer.
- Storing text in registers — Tells how to store and retrieve text from a register.

You can view the kill history by executing the Show Kill History command (CTRL-STATUS).

The following list defines quantities on which the transpose, mark, and delete commands operate:

- Character — A letter, number, or special symbol, such as a punctuation character.
- Line — A line is ended by a return character. A line can span multiple lines on the screen before the return character appears. (Zmacs recognizes return characters, but they are not displayed in the buffer.)
- Word — A group of contiguous letters with no blanks.
- Symbolic expression — Lisp expressions enclosed in parentheses or a single symbol, such as **my-symbol**.
- Sentence — A sentence is ended by a period (.), question mark (?), or exclamation point (!) followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote ("), single apostrophe ('), closing parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.
- Definition — Any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **deffavor**, **defmethod**, and **defvar**.
- Paragraph — A paragraph is delimited by blank lines or by lines that start with a delimiter in the Zmacs user variable Paragraph Delimiter List or in the user variable Page Delimiter List. You can see and change the values of these variables by executing the Variable Apropos command (HELP V).
- Region — A quantity that you define by marking.

Exchanging Text

2.13.1 Zmacs allows you to exchange various quantities of text. When you *exchange* text, you transpose the positions of two items of that quantity. For example, if you exchange the characters *ex*, the result is *xe*. You can exchange the following quantities of text:

- Characters
- Lines
- Words
- Symbolic expressions
- Paragraphs

The following commands allow you to exchange text:

- Exchange Characters (CTRL-T) — Exchanges the characters before and after point (that is, the character before the keyboard cursor and the character the cursor is on).
- Exchange Lines (CTRL-X CTRL-T) — Exchanges the line the keyboard cursor is on with the line above the cursor.
- Exchange Sexps (META-CTRL-T) — Exchanges the symbolic expression the keyboard cursor is on with the expression before the cursor.
- Exchange Words (META-T) — Exchanges the word the keyboard cursor is on with the word before the cursor.
- Various Quantities (CTRL-Q) — Exchanges various quantities, such as character, line, symbolic expression, word, paragraph, and so on. See paragraph 3.4.5, Various Quantities Command, for more information.

If you want more information on these commands, refer to the Text Format commands in Section 3, Command Groups.

Marking Text

2.13.2 You mark text so that you can manipulate it as one block. The marked text is highlighted (usually by underlining) and called a *region*. You can use commands that mark text in quantities such as word, symbolic expression, or paragraph; or you can decide the size of the text you want to mark. You can mark a region as small as one character or as large as the entire buffer. Some of the uses for marking text are as follows:

- To put the region on the kill history by killing or saving it
- To see how large a Lisp expression is
- To evaluate or compile the region
- To save the region in a register
- To change the font of the region
- To add semicolons (;) in front of every line in the region

- To remove semicolons from lines in the region
- To fill the text in the region
- To indent the region
- To lowercase or uppercase the region
- To print the region

You can execute the Apropos command (HELP A) and type `region` to see the commands that deal with regions.

When you mark text, you are setting the buffer pointers in the editor at either end of the block of text. One of them is called *point*, which is where the keyboard cursor is. The other is called *mark*, which is the buffer pointer in the editor that allows you to mark text. The text between point and mark is highlighted (usually by underlining) regardless of whether point or mark comes first.

The following paragraphs describe how to mark (and unmark) text of various sizes.

Mark a Region Zmacs provides two procedures for marking a region. When you mark a region, you decide the size of the region. One procedure is as follows:

1. Press the left mouse button and hold it down.
2. Move the mouse cursor until you have marked all the text you want.
3. Release the left button.

Another procedure is as follows:

1. Set mark at point with the Set Pop Mark command (CTRL-space bar).
2. Move the keyboard cursor with one of the keyboard cursor movement commands. (Do not use the mouse to move the cursor.) The area between mark and the cursor is marked.

Unmark a Region To unmark a region, you press CTRL-G. This command is the Abort One Level (Unmark Region) command.

Mark a Symbolic Expression Zmacs provides two procedures for marking a symbolic expression. One procedure is as follows:

1. Position the mouse cursor on a symbolic expression.
2. Click middle to mark it.

Another procedure is as follows:

1. Position the keyboard cursor on the first character of a symbolic expression.
2. Press META-CTRL-@. (This command is the Mark Sexp command.) To mark more than one symbolic expression from point, you supply a numeric argument.

Mark a Word Zmacs provides two procedures for marking a word. One procedure is as follows:

1. Position the keyboard cursor on the first character of a word.
2. Press META-@. (This command is the Mark Word command.) To mark more than one word from point, you supply a numeric argument.

Another procedure is as follows:

1. Position the mouse cursor on a word.
2. Click middle to mark it.

Mark a Definition Zmacs provides two procedures for marking a definition. One procedure is as follows:

1. Position the keyboard cursor on a definition such as a **defun**.
2. Press META-CTRL-H. (This command is the Mark Definition command.)

Another procedure is as follows:

1. Position the mouse cursor on the opening parenthesis of a definition such as a **defun**.
2. Click middle to mark it.

Mark a Paragraph You can use the following procedure to mark a paragraph:

1. Position the keyboard cursor on a paragraph.
2. Press META-H. (This command is the Mark Paragraph command.)

Let Zmacs Mark Text for You You can use the following procedure to let Zmacs determine the length of what you want marked:

1. Position the mouse cursor on the item (such as a Lisp expression) you want marked.
2. Click middle. (This command is the Mark Item command.) Zmacs determines where to put point and mark in order to include the item you indicate.

If you click middle on the opening parenthesis of a Lisp expression, Zmacs marks the entire Lisp expression; that is, Zmacs marks all the way to the corresponding right parenthesis. The corresponding right parenthesis may not

appear for several lines. You can mark an entire definition by clicking middle on the opening (or closing) parenthesis of the definition.

Deleting and Killing Text

2.13.3 Zmacs contains many commands to delete and kill text in your buffer. When you *delete* text, you cannot recover it. Usually, you delete only small items such as characters. When you *kill* text, you delete it from your buffer and put it on the kill history. You can yank (copy) the text back into your buffer at a later time. You can also *save* text on the kill history without deleting it from your buffer.

The size of the text you kill can be as small as a single character or as large as the entire buffer. You decide the length of the text you want to kill by first marking the text as a region. Zmacs also provides commands to kill quantities such as words, symbolic expressions, and lines.

The following paragraphs describe the procedures to delete, kill, and save text.

Deleting Characters You can enter the following commands to delete characters:

- Delete Forward (CTRL-D) — Deletes one or more characters forward from point.
- Delete Backward (RUBOUT) — Deletes one or more characters backward from point.

When you delete one character, it is not put on the kill history. However, if you delete more than one character at a time by supplying a numeric argument to CTRL-D or RUBOUT, these commands become kill commands and what they delete is put on the kill history.

Killing Words You can use the following commands to kill words:

- Backward Kill Word (META-RUBOUT) — Kills one or more words backward from point.
- Kill Word (META-D) — Kills one or more words forward from point.

To kill more than one word at a time, you supply a numeric argument with these commands.

Killing Symbolic Expressions You can enter one of the following commands to kill symbolic expressions:

- Backward Kill Sexp (META-CTRL-RUBOUT) — Kills one or more symbolic expressions backward from point.
- Kill Sexp (META-CTRL-K) — Kills one or more symbolic expressions forward from point.

To kill more than one symbolic expression at a time, you supply a numeric argument with these commands.

Killing Sentences You can use the following commands to kill sentences:

- **Backward Kill Sentence (CTRL-X RUBOUT)** — Kills one or more sentences backward from point.
- **Kill Sentence (META-K)** — Kills one or more sentences forward from point.

To kill more than one sentence at a time, you supply a numeric argument with these commands.

Killing Lines To kill a line, you can use the Kill Line (CTRL-K) command to kill text to the end of the line. If only blanks and a return character are to the right of the cursor, this command kills them.

If the keyboard cursor is in the middle of a line when you kill the line, Zmacs kills from the cursor to the end of the line. However, it leaves a return character at the end of the line.

If you want to kill the whole line, you need to move to the beginning of the line. Zmacs kills all the contents of the line but does not touch the return character. Thus, you have a blank line there. If you kill the line again, Zmacs deletes the return character on that line and the line disappears.

When you kill lines consecutively, Zmacs puts them together in one entry in the kill history. You can execute the Show Kill History command (CTRL-STATUS) to yank individual lines with the mouse. If you do not want the killed lines together as one entry in the kill history, you need to move the keyboard cursor or execute another command in between each Kill Line command.

Killing and Saving Regions You can kill or save regions of any length. You decide the length of the region when you mark it. The following procedure describes how to save a region:

1. Mark a region.
2. Double click middle (or press META-W). Zmacs saves the region you marked in the kill history without removing it from your buffer. (This is the Save Region command.)

When you save a region, notice that the highlighting (usually underlining) disappears. However, the region is still remembered; if you double click middle again without moving the keyboard cursor, the region is deleted from your buffer. (This command is the Kill Region command.)

NOTE: If you move the cursor or perform another operation between double clicks, the region is not deleted. Zmacs retrieves (yanks) the region from the kill history into your buffer on the second double click.

The following procedure describes how to kill a region:

1. Mark a region.

2. Double click middle twice (or press CTRL-W). Zmacs puts the region you marked on the kill history and deletes it from your buffer. (This command is the Kill Region command.)

Retrieving (Yanking) Text

2.13.4 You can retrieve (yank) any entry from the kill history and put it in your buffer. You can also yank individual lines from the entry. To copy blocks of text in Zmacs, execute a save followed by a yank. To move blocks of text in Zmacs, execute a kill followed by a yank.

Zmacs provides several procedures for yanking entries from the kill history:

- Mouse clicks (double middle) — Allows you to yank the top entry on the kill history, yank more than one copy of the entry, or rotate through the kill history.
- Show Kill History command (CTRL-STATUS) — Displays the kill history and allows you to yank any entry by using the mouse. You can yank as many copies of an entry as you want. You can also yank individual lines.
- Yank command (CTRL-Y) — Allows you to yank the top entry of the kill history and yank more than one copy of the entry.
- Yank Pop command (META-Y) — After you use the Yank command (CTRL-Y) once, you can use META-Y to rotate through the kill history.

When you yank an entry (or an individual line) from the kill history, Zmacs inserts it where the cursor is. The cursor is then positioned at the end of the inserted text.

The Show Kill History command allows you to view the kill history and yank entries from it. Figure 2-8 shows an example kill history display.

Figure 2-8 Show Kill History Display

```

*Mouseable Kill Ring Contents:
  Insert a Carriage Return.          Toggle of Insert a Carriage Return.
  Finished                            Toggle finished when entire yank is done.

1:<< ALL of the following kill. >>
;close the output stream
  (close sunstrn)
  ; (beep)
  (send tv:selected-window :beep :chime)
)

2:<< ALL of the following kill. >>
;make an array called invert-pix-array that matches the properties of pix-array,
;it will contain the bit map formatted for the Interleaf.
  (setf invert-pix-array (make-array '(832 1024) :type 'art-1b))

3:<< ALL of the following kill. >>
(DEFUN PLOT-LINE (length angle)
  (declare (special x y))
  (connect-line (+ x (* length (cos angle)))
               (+ y (* length (sin angle)))))

```

You yank entries from this display by clicking on <<All of the Following kill.>>. This selection yanks the entire entry. You can also yank individual lines by clicking on them.

Anytime you yank an entry with the mouse, Zmacs inserts a return character in your buffer before the entry. If you do not want to insert a return character, you can click on the `Toggle of Insert a Carriage Return` prompt in the upper right corner of the display. If you want to remove the kill history display automatically after you yank an entry, you can click on `Toggle finished when entire yank is done`.

When you finish, you can remove the kill history screen by pressing the space bar or by clicking on `Finished`. If you want more information on the `Show Kill History` command, refer to the `Deleting and Moving Text` commands in Section 3, `Command Groups`.

The following list describes the procedures for yanking kill history entries by using the mouse, the `Show Kill History` command (`CTRL-STATUS`), the `Yank` command (`CTRL-Y`), or the `Yank Pop` command (`META-Y`):

- **Yank the top or most current entry on the kill history:**
 - With the `Show Kill History` command (`CTRL-STATUS`), click on `<<All of the following kill.>>`.
 - With the `Yank` command, press `CTRL-Y`.
 - With the mouse, double click middle.
- **Yank an entry more than once:**
 - With the `Show Kill History` command (`CTRL-STATUS`), click on `<<All of the following kill.>>` twice. You can yank any entry you want. Clicking once may remove the display (see the mode line window contents). You can fix this removal by clicking on the option in the upper right corner of the display called `Toggle finished when entire yank is done`.
 - With the `Yank` command, press `CTRL-Y` twice. You can yank only the top entry on the kill history.
 - With the mouse, double click middle, move the cursor, and then double click middle again. You can yank only the top entry on the kill history.
- **Yank any entry on the kill history:**
 - With the `Show Kill History` command (`CTRL-STATUS`), click on `<<All of the following kill.>>`, which precedes any entry.
 - With the `Yank` command (`CTRL-Y`), supply a numeric argument. For example, if you want the third entry on the kill history, enter `CTRL-3 CTRL-Y`.
 - With the `Yank` command (`CTRL-Y`) and the `Yank Pop` command (`META-Y`), press `CTRL-Y` for the first entry and then `META-Y` for each successive entry. This procedure allows you to rotate through the kill history.
- **To yank individual lines, use the `Show Kill History` command (`CTRL-STATUS`), and select individual lines with the mouse.**

Storing Text in Registers

2.13.5 Zmacs contains storage areas called *registers* in which you can store either text or cursor locations. You can even store text and a cursor location simultaneously in one register. The text that you store can be of any size. You can store only one block of text and one cursor location in a register. However, they stay in a fixed place. On the kill history, text is moved down the stack as new entries are added. (Paragraph 2.11.3.2, Storing Cursor Locations in Registers, discusses storing cursor locations in registers.)

Registers have one-letter names. For example, you can have a register named A, B, C, or even \$, as long as the name is a single character. You cannot specify a character with attribute bits (that is, a key used in combination with SUPER, HYPER, META, or CTRL). The limit on the number of registers you can create is the number of characters available to name them.

When you retrieve text from a register, Zmacs inserts the text where the cursor is. The cursor is then positioned at the end of the inserted text.

The following paragraphs describe how to store and retrieve text from a register and also how to view a register.

Storing Text in a Register Use the following procedure to store text in a register:

1. Mark a region.
2. Press CTRL-X X. (This command is the Put Register command.)
3. Type any character, such as A, to specify a register.

NOTE: You can store a block of text and a cursor location simultaneously in the same register. You use the Put Register command (CTRL-X X) to store the text and the Save Position command (CTRL-X S) to store the cursor location. For both commands, you specify the same register name.

Retrieving Text From a Register You can use the following procedures to yank text from a register into your buffer. The first procedure allows you to view one register and yank text from it:

1. Press META-X and type `view Register`.
2. Type the name of the register you want (for example, A).
3. Click on the line that says `<<Yank the entire register >>`. You can also yank individual lines by clicking on them.

The following procedure allows you to view all registers and yank text from them:

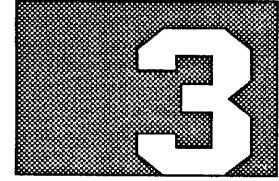
1. Press META-X and type `Show All Registers`.
2. Click on the line that says `<<Yank the entire register >>`. You can also yank individual lines by clicking on them.

The following procedure allows you to yank text from a register. It does not display the contents of the register:

1. Press CTRL-X G. (This command is the Open Get Register command.)
2. Type the name of the register you want (for example, A).

Viewing Registers You can use the following commands to view registers:

- **View Register (META-X View Register)** — Displays the contents of the register you specify. The display contains both text and cursor locations, and it is mouse-sensitive. You can yank the text in its entirety or on a line-by-line basis.
- **Show All Registers (META-X Show All Registers)** — Displays the contents of all registers. The display contains only text (not the cursor locations), and it is mouse-sensitive. You can yank the text in its entirety or on a line-by-line basis.
- **List Registers (META-X List Registers)** — Displays the contents of all registers. The display contains both text and cursor locations, but it is not mouse-sensitive.



COMMAND GROUPS

Overview of Commands

3.1 This section describes all the Zmacs commands. These commands are divided into functional groups to explain how the commands work together and to help you easily find the particular command you need. Each individual command description tells in detail what the command does and lists the keystroke that executes it. The following list describes the command groups:

- Buffer commands — Perform operations on buffers such as edit, list, kill, and print.
- Compile and Evaluate commands — Allow you to compile and evaluate files, buffers, regions, and changed definitions from within Zmacs.
- Cursor Movement commands — Provide a wide range of options to move the cursor and allow you to store cursor locations.
- Customization commands — Customize some of the operations of Zmacs by changing Zmacs user variables, putting a series of commands on one key, and so on.
- Deleting and Moving Text commands — Allow you to delete, move, copy, and mark text.
- Directory commands — Allow you to perform operations such as creating and editing a directory.
- File commands — Include such operations as find, copy, load, write, and save files.
- Font commands — Include commands that you can use in your buffer, such as set, change, and display fonts.
- Help, Documentation, and Undo commands — Tell how to abort Zmacs operations, receive help on what you are doing, and undo previous commands.
- Lisp Programming commands — Help you develop your Lisp program with facilities such as editing Lisp definitions, listing callers, editing flavors, and comparing and merging two files or buffers.
- Lisp Syntax commands — Work on the syntax of Lisp code by manipulating comments and parentheses, expanding macros, and grinding (pretty printing).
- Miscellaneous commands — Count quantities such as words and insert the date into your buffer.
- Mode and Buffer Attribute commands — Set major and minor modes and other buffer attributes such as package and base.

- Mouse commands — Allow you to perform operations with the mouse, such as moving the keyboard cursor to the mouse cursor.
- Print commands — Allow you to print on a hard-copy device and to grind (pretty print) Lisp code.
- Scroll commands — Allow you to scroll editor buffer windows in Zmacs.
- Search and Replace commands — Perform elaborate search and replace functions and let you specify a group of files to be treated like one file for searching and replacing.
- Text Format commands — Allow you to exchange (transpose) text, fill text, lowercase and uppercase text, and indent Lisp code.
- Window commands — Allow you to split the editor buffer window into two or more windows.

The following topics relate to your use of all the Zmacs commands. These topics are described in detail in the paragraphs that follow.

- Executing commands — Describes the META-X and META-CTRL-X commands, which allow you to execute commands by their name.
- Prefix command — Describes the CTRL-X command, which is a prefix for a number of other commands.
- Minibuffer commands — Details the important commands that you can use in the minibuffer to complete command names and pathnames, display a menu of possible completions, and repeat the previous minibuffer command.
- Numeric arguments — Tells you how to enter numeric arguments to Zmacs commands.

Executing Commands

3.1.1 Zmacs allows you to execute commands by keystrokes, by clicking the mouse on command names in menus, and by typing the command name after pressing META-X or META-CTRL-X. Many Zmacs commands do not have keystrokes assigned to them. These commands are called *extended commands*. The following procedure describes how to execute one of these commands:

1. Press META-X.

A prompt appears in the minibuffer asking you for a command name.

2. Enter the name of the command; for example, type `split screen`.

Refer to the Minibuffer commands, paragraph 3.1.3, for information on the online help available for finding and completing command names. Zmacs automatically completes a command name for you if you supply enough characters to uniquely identify the command name and then press ESCAPE.

The following table describes META-X and META-CTRL-X in more detail:

Keystroke	Description
META-X	Executes any legal Zmacs commands that Zmacs makes available through their names. Commands allowed by META-X are designed to apply to your current context in the editor. Zmacs is extendable: you can define your own commands and add them to a command table.
META-CTRL-X	Executes any legal Zmacs commands that Zmacs makes available through their names. META-CTRL-X does not guarantee that these commands apply to your current context in the editor.

CAUTION: Since META-CTRL-X can execute commands out of context, problems may arise. You should not use META-CTRL-X except for trying out new commands.

Prefix Command 3.1.2 The Prefix command (CTRL-X) allows you to use a group of commands after you press CTRL-X. That is, you then press another keystroke after CTRL-X to execute a particular command. For example, to execute the Write File command, you press CTRL-X CTRL-W. Many Zmacs commands use CTRL-X as part of their keystrokes.

If you press CTRL-X and decide you want to abort, you can press ABORT or CTRL-G.

Minibuffer Commands 3.1.3 Zmacs uses the minibuffer for a wide range of operations. You enter command names to META-X and responses to command prompts in the minibuffer. You can tell if you are in the minibuffer because the keyboard cursor moves there.

The minibuffer is usually a three-line-high version of the editor. It works exactly like an editor buffer window. You can edit your input with the cursor movement commands and the delete commands. You can also scroll in it. One big difference is that you cannot enter the minibuffer from the minibuffer; that is, you can enter the minibuffer from the editor buffer window, but you cannot enter another minibuffer from the minibuffer.

When you enter the minibuffer, the right side of the mode line presents messages about the operation you are performing. The following messages may appear:

- **Completion** — If you press the ESCAPE key when you see this message, Zmacs provides help on completing command names, pathnames, and responses to commands.
- **Extended Search Characters: CTRL-H** — If you press CTRL-H HELP when you see this message, a list of extended search characters appears. You use these to perform complicated searches.

The following table describes important keystrokes that you can use in the minibuffer:

Keystroke	Description
HELP	Provides help on the operation you are performing.
ESCAPE	Completes the pathname as much as possible but does not execute it. You can use this keystroke to verify that Zmacs completes your partial input the way you expect. For example, the Find File command creates a new buffer if you type a wrong partial pathname and press RETURN; to avoid this, press ESCAPE to determine if the command completes the pathname as you intended.
END	Attempts completion on the partial input and exits if it is unique. Note that although a completion is unique, it is not necessarily what you had in mind.
RETURN	Usually exits without completion. Sometimes RETURN completes, such as for command names. Also, for commands that allow you to type multiple lines, such as the Evaluate Minibuffer command (META-ESCAPE), RETURN only inserts return characters, and END exits.
CTRL-G	Clears the minibuffer if there is text in it. If the minibuffer is empty, CTRL-G quits the minibuffer.
CTRL-SHIFT-Y	During execution of one of the File commands, inserts the default pathname as text.
META-SHIFT-Y	During execution of one of the File commands, inserts the last pathname you typed, as text.
CTRL-/	Performs the Apropos command for the possible completions of what you have typed so far. (You type part of a command name and CTRL-/ searches for the possible completions.) CTRL-/ looks for any matching string regardless of its position in the command name. Also, the display it provides is mouse-sensitive. You can position the mouse cursor on a command name and click left to execute it.
CTRL-?	Gives a menu of the possible completions for the string that you have typed so far. Unlike CTRL-/, CTRL-? looks only for matching prefix strings (that is, it only finds the commands that begin with the string). The display that CTRL-? provides is mouse-sensitive. You can position the mouse cursor on a command name and click left to execute it.

Keystroke	Description
META-CTRL-Y	Cycles backward through previous minibuffer commands and the data they were given when entered.
CTRL-SHIFT-F	Specifies a pathname instead of a buffer name. You use this keystroke when a file is mismatched with its buffer. For example, you may create a buffer and read a file into the buffer without the buffer name being updated.

CTRL-X ESCAPE is an important keystroke that allows you to repeat a minibuffer command. This command does not work if you are already in the minibuffer.

CTRL-X ESCAPE with a numeric argument executes the n th previous command, where n is the numeric argument. The default is 1. An argument of 0 lists the commands that are remembered.

Numeric Arguments

3.1.4 You can give any Zmacs command a numeric argument. The numeric argument has different effects depending on the command. You can supply positive or negative numbers.

One of the most common uses for the numeric argument is as a counter for the number of times the command should be performed. For example, supplying Forward Character (CTRL-F) a numeric argument of 20 moves the cursor forward 20 characters instead of 1.

Some commands only need to know if there is a numeric argument present; they do not care about its value. For example, the Evaluate Buffer Changed Definitions command evaluates any definitions in a buffer that you have edited. If you supply a numeric argument with the command, it asks about each definition individually.

Other commands react differently to different numeric arguments. An example is the Select Previous Buffer (META-CTRL-L) command. If you supply a numeric argument, it selects the n th previous buffer. An argument of 1 rotates through the buffer history. Any negative argument rotates through the buffer history in the opposite direction.

The Copy File command provides another example. Each of the numeric arguments 2 through 6 causes a different operation to be performed.

The individual documentation for each command tells you how numeric arguments affect the command.

To specify a numeric argument for a command, you hold down any combination of the CTRL, META, SUPER, and HYPER keys while pressing the desired numbers. Then you type the command. For example, if you want to move the cursor forward 20 characters, you press CTRL-2 CTRL-0 CTRL-F.

To enter a numeric argument for a META-X command, you press the appropriate keys for the numeric argument such as CTRL-2, then press META-X, and finally type the name of the command. For example, if you want to specify a numeric argument of 2 for the Copy File command, you press CTRL-2 META-X and then type `copy File`.

NOTE: This manual expresses numeric arguments with the convention of typing only one digit at a time for each CTRL. Thus, a numeric argument of 20 is written CTRL-2 CTRL-0. However, you can hold down the CTRL key until you finish entering all of the numbers.

The Universal Argument command (CTRL-U) provides another way of entering numeric arguments. If you enter this command by itself, it multiplies the current argument by four. (The current argument is initially 1.) For example, CTRL-U CTRL-F moves the cursor forward four characters. If you press CTRL-U twice, the argument is multiplied by 16. Thus, CTRL-U CTRL-U CTRL-F moves the cursor forward 16 characters. CTRL-3 CTRL-U CTRL-F moves the cursor forward 12 characters.

Other commands treat CTRL-U differently. Commands that deal with packages search the current package without a CTRL-U argument, search all packages with one CTRL-U, or ask for a package if you press CTRL-U twice. Refer to paragraph 3.11.2, Caller Commands.

**Buffer
Commands**

3.2 The Buffer commands allow you to perform a multitude of operations on a buffer. The commands are divided into the following groups:

- Capture Into Buffer commands — Insert the results of Lisp functions into your buffer.
- Insert Buffer commands — Append to and insert buffers.
- Kill and Save Buffers commands — Kill and save buffers.
- List and Edit Buffers commands — Allow you to list buffers and perform a variety of operations on them, such as editing, saving, writing, compiling, killing, and printing.
- List and Edit Changed Definitions commands — List sections (definitions) in a buffer and list and edit modified definitions.
- Miscellaneous Buffer commands — Perform operations on buffers such as renaming, viewing, sectionizing, and listing modifications.
- Print Buffer commands — Print buffers.
- Revert Buffer commands — Removes the changes you made to a buffer.
- Select Buffer commands — Allow you to switch between the buffers you are editing.

NOTE: For information on the commands that change buffer attributes such as mode, package, base, fonts, and tab width, refer to the Mode and Buffer Attribute commands.

The List Buffers command (paragraph 3.2.4) and the Kill or Save Buffers command (paragraph 3.2.3) display menus that allow you to perform a wide range of operations on buffers, such as editing, compiling, killing, printing, saving, and writing. These two commands provide a tool for dealing with buffers.

**Capture Into
Buffer Commands**

3.2.1 These commands allow you to insert the results of Lisp functions into the current buffer.

Evaluate and Replace Into Buffer

Command

Keystroke: META-X Evaluate and Replace Into Buffer

Evaluates the symbolic expression after point and places the result into the buffer at point. The original expression is deleted, and the value that is printed out replaces it.

Also refer to the Compile and Evaluate commands.

Evaluate Into Buffer

Command

Keystroke: META-X Evaluate Into Buffer

Evaluates a form from the minibuffer and inserts the results into the buffer at point. If the form returns multiple values, each value is printed into the buffer with a carriage return before each one. If you supply a numeric argument with this command, output printed by the evaluation also goes in the buffer.

Also refer to the Compile and Evaluate commands.

Execute Command Into Buffer

Command

Keystroke: META-X Execute Command Into Buffer

Executes the next editor command, printing the result into the buffer. Any output that ordinarily appears as typeout from the command is inserted into the current buffer instead. Trace and warning output are also inserted into the buffer.

**Insert Buffer
Commands**

3.2.2 The Insert Buffer commands allow you to append to and insert buffers.

Append to Buffer

Command

Keystroke: CTRL-X A

Appends a region to the specified buffer. You are prompted in the minibuffer for the name of the buffer; it is created if it does not already exist. This command inserts the text at that buffer's point and leaves point after the inserted text. With a numeric argument, this command leaves point before the inserted text.

Insert Buffer

Command

Keystroke: META-X Insert Buffer

Inserts a copy of the specified buffer at point.

**Kill and Save
Buffer Commands**

3.2.3 These commands allow you to kill and save buffers. One of the commands, the Kill or Save Buffers command, presents a listing of the Zmacs buffers and allows you to compile or unmodify them in addition to saving and killing them.

Kill Buffer

Command

Keystroke: CTRL-X K

Kills a specified buffer. You are prompted in the minibuffer for the name of the buffer to kill.

Kill or Save Buffers

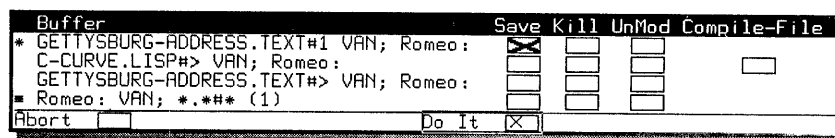
Command

Keystroke: META-X Kill or Save Buffers (also available on the Editor menu)

Displays a menu containing the Zmacs buffers and allows you to save buffers to disk, kill buffers, unmodify buffers, and compile them if they are Lisp type.

The Kill or Save Buffers menu is a *multiple-choice menu* in which you have to select the Do It or Abort item with the mouse to remove the menu. Figure 3-1 shows a sample Kill or Save Buffers menu.

Figure 3-1

Kill or Save Buffers Menu

The following table explains the symbols in the left column:

Symbol	Description
*	The buffer has been modified.
≡	Read only.
+	New file (not yet written to disk).

Some of the boxes are marked with an x, such as files to be saved. If you modify the buffer and it is connected to a source file on disk, an x appears in the Save box. An x does not appear in the Save box for a new buffer (not connected to a file) that you modify. In this case, only an asterisk (*) appears in the left column.

When you move the mouse over a box, the mouse cursor becomes an x. If you click left, an x appears in the box to select whatever the box represents. If you click left again while the mouse cursor is still an x, the x in the box is removed. You can choose to both save and kill a buffer. No actions are taken until you click on the Do It item.

If you select the Do It item, the operations are performed for the boxes that have an x in them. For example, files that are marked to be saved are written to disk. Files that are marked to be killed are killed. If you select the Abort item, no operation is performed, regardless of what is marked.

When you invoke the Kill or Save Buffers menu again, the items from which you removed the x are marked again. However, the Unmod option removes the x.

The following paragraphs explain in detail the options on the Kill or Save Buffers menu.

Save The Save option writes the buffer to disk, using the pathname of the file from which the buffer originally came and incrementing the version number. This option automatically provides the default file pathname; you do not need to specify the filename. If you want to specify a different filename, you must use the Write option on the List Buffers menu. If the buffer is a new buffer not associated with a pathname, you are prompted for a filename.

Kill The Kill option deletes the buffer.

Unmod The Unmod option marks the buffer as unmodified. (It does not discard the changes.) Buffers that you have modified are marked with an *x* in the Save box. The normal way to end an edit session is to click on the Do It item and save all modified files. However, if you make modifications to files that you do not want to save, you must toggle the *x* out of the Save box each time you invoke the Kill or Save Buffers menu. This procedure is not necessary if you mark the Unmod box (and unmark the Save box) for the buffer. The *x* does not reappear in the Save box until you change the buffer again.

Compile The Compile option compiles the file that the buffer is connected to on disk and puts the compiled version on disk with the same filename and a different type, *xld*. It does not change the memory from which you execute. If the Save box also has an *x* in it, the buffer is written to disk before it is compiled.

A box does not appear in the Compile column for a buffer unless the buffer is Lisp type. Also, a box does not appear unless there is a source file on disk. You must save a new buffer (Lisp type) to disk before a box appears in the Compile column.

The Compile option performs the same operation as the Compile File command (META-X Compile File).

Kill Some Buffers

Command

Keystroke: META-X Kill Some Buffers

Offers to kill each buffer. For each buffer, this command asks whether to kill it, and for each one to be killed, it offers to write out any changes.

List and Edit Buffer Commands

3.2.4 Zmacs provides two commands, List Buffers and Edit Buffers, that display the list of Zmacs buffers and allow you to perform a variety of operations on the buffers. Between the two commands, you can select buffers for editing and perform other operations such as compiling, printing, killing, saving, writing, and reverting. The Edit Buffers command allows you to perform operations on all the buffers at one time. The List Buffers command works on one buffer at a time.

Buffer Edit

Command

Keystroke: META-X Buffer Edit

Edits the list of buffers; saves, kills, and so on. (This command is a synonym for the Edit Buffers command.)

Edit Buffers

Command

Keystroke: META-X Edit Buffers

Edits the list of Zmacs buffers. Performs operations such as select, save, write, kill, and unmodify buffers. The Edit Buffers command allows you to specify one of these commands by typing the character for the command next to the buffer name.

When you enter one of the Edit Buffers commands, the keyboard cursor must be on the line that contains the name of the buffer for which you want to execute one of the commands. A mark appears in the first column signifying the operation (for example, S for save). You can mark different buffers for different operations. When you finish marking buffers, you press Q and the operations are performed.

NOTE: If the buffer is not associated with a file on disk, many of these commands take effect immediately. In the case of S (for save), you are prompted in the minibuffer for a filename.

Most of the cursor movement commands work the same way in the Edit Buffers display as they do in the editor buffer. You can use the following methods to move the cursor:

- CTRL-P and ↑ — Take you to the previous line.
- Space bar, CTRL-N, and ↓ — Take you to the next line.
- Mouse — Positioning the mouse cursor beside a filename and clicking left moves the keyboard cursor there.

When you go to another buffer (for example, by typing a period to select a buffer), you can return to the Edit Buffers display by pressing META-CTRL-L. (META-CTRL-L is the Select Previous Buffer command.) You can also return to the Edit Buffers display by executing the List Buffers command (CTRL-X CTRL-B). The buffer name containing the Edit Buffers display is then listed.

The following table describes the Edit Buffer commands. Pressing the HELP key provides an online description of the Edit Buffer commands.

Keystroke	Description
HELP	Provides a brief description of the Edit Buffer commands.
D or K	Marks the buffer to be killed. Also requests saving, if the buffer contains changes. Use N to cancel the saving but not cancel the killing.
U	Cancels all operations on the buffer.
RUBOUT	Moves to the previous line and cancels all operations on that line.
.	Marks this buffer to be selected.
P	Marks this buffer to be printed.
S	Marks this buffer to be saved. The buffer is saved to the file (on disk) from which it originally came. If you want to specify a different file or if it is a new buffer, use the W command.
W	Marks this buffer to be written. The buffer is written to a file (on disk) that you specify.
~	Marks this buffer to be unmodified.
R	Marks this buffer to be reverted; that is, the changes you made are forgotten.
N	Cancels any request for file I/O on the buffer, such as save, revert, and so on.
Q	Exits Edit Buffers. Kills, saves, and reverts as requested. Displays the files to be deleted and/or printed, then asks you to confirm.
ABORT	Aborts Edit Buffers. If you marked a file for operation (such as D for deletion), the ABORT key leaves Edit Buffers and usually cancels the operations. However, if the buffers need saving, for example, the marks stay.
END	Exits Edit Buffers. Kills, saves, and reverts as requested. Displays the files to be deleted and/or printed, then asks you to confirm.

Figure 3-3 List Buffers Menu

Buffer name:	File Version:	Major mode:	DRAW-ICON.MAGIC#> WEBB.WINDOW; Lima:
DRAW-ICON.MAGIC#> WEBB.WINDOW; Lima:	(1)	(Common-Lisp)	Compile File Kill
ICONS-EXPLAN.LISP#> WEBB.WINDOW; Lima:	(1)	(Common-Lisp)	Print Unmod
BINDINGS.EXAMPLE#> WEBB.WINDOW; Lima:	(3)	(Common-Lisp)	Save Write
Possibilities Lists	[7 Lines]	(Possibilities)	Select
Definitions	[4 Lines]	(Possibilities)	
mh:laurie; modified-file	[1 Line]	(Common-Lisp)	
SHEET.LISP#> WINDOW; MR-X:	(12)	(Common-Lisp)	
mh:narty;new-file.example	[1 Line]	(Common-Lisp)	
Lima: WEBB; *.*#* (1)	[33 Lines]	(Direc)	
Buffer-3	[1 Line]	(Common-Lisp)	
Buffer-1	[1 Line]	(Common-Lisp)	
* means buffer modified. ■ means read-only.			
(END-ITEM-LIST (AREF ROW=MAP (1+ ROW)))			
(STR)			
(FONT)			
(FLAG)			
(X-POS 0)			

The following paragraphs explain in detail each of the options on the List Buffers menu.

Compile File The Compile File option compiles the file that the buffer is connected to on disk and puts the compiled version on disk with the same filename and a different type, xld. It does not change the memory from which you execute. If you have made changes to the buffer, you are first asked if you want to save the buffer so that your changes will be in the file on disk.

NOTE: The Compile File option works only if there is a source file on disk. You must save a new buffer to disk before using the Compile File option.

The Compile File option performs the same operation as the Compile File command (META-X Compile File).

Kill The Kill option deletes the buffer. If the buffer is connected to a file, you are asked if you want to save the buffer first. If you do not save it, the original version on disk is unchanged.

Print The Print option automatically prints a copy of the buffer to a line printer. (This option may not be functional at some sites.)

Unmod The Unmod option marks the buffer as unmodified so that you can direct the editor to consider the buffer unchanged. (It does not discard the changes.) The asterisk (*) that you see in the left column of the buffer history and the mode line disappears. This option is useful when you execute the Kill or Save Buffers command.

NOTE: If you invoke the Kill or Save Buffers menu from the Top-Level Editor menu, you see a multiple-choice menu. Files that you have modified are marked with an x in the Save box. The normal way to end an edit session is to click on the Do It item to save all modified files. However, if you make modifications to files that you do not want to save, you must toggle the x out of the Save box each time you invoke the Kill or Save Buffers menu. This procedure is not necessary if you unmodify the buffer. The x does not reappear in the Save box until you change the buffer again.

Save The Save option writes the buffer to disk, using the pathname of the file from which the buffer originally came and incrementing the version number. This option automatically provides the default file pathname. Thus, you do not need to specify the filename. If you want to specify a different filename, you must use the Write option.

Write The Write option writes the buffer to a file that you specify. It is similar to the Save option except that you are allowed to specify the filename rather than taking the default. If you accept the default, this option works like Save.

Select The Select option allows you to edit the buffer.

List and Edit Changed Definition Commands

3.2.5 These commands allow you to perform operations on the changed definitions in a buffer. When Zmacs reads into a buffer a file that contains Lisp code, it divides the definitions in the buffer into sections. The only buffers that have sections are those containing Lisp code. Other buffers do not have sections. A *section* can contain any defining construct that begins with `def` after the opening parenthesis in the first column, such as `defvar`, `defflower`, and `defun`. The defining construct can be user-defined. Each definition is a section.

At the top of the buffer is a section called the *buffer header*, which contains the attribute list.

For a detailed discussion of this section scheme, refer to paragraph 2.10.4, Lisp Code in Buffers.

Basically, the section scheme allows you to locate definitions quickly and perform operations only on the changed definitions. The List and Edit Changed Definitions commands allow you to perform the following operations:

- List all the sections (definitions) in a specified buffer.
- List or edit the definitions that have been changed in a specified buffer or in all buffers.

You can compile and evaluate only the changed definitions in a buffer. Refer to the Compile and Evaluate commands.

For all of these commands except the List Sections command, you can press CTRL-SHIFT-P to start editing the changed definitions. Each successive CTRL-SHIFT-P moves the cursor to the next changed definition. However, you cannot return to the beginning of the list.

Edit Buffer Changed Definitions

Command

Keystroke: META-X Edit Buffer Changed Definitions

Edits any definitions in the buffer that have been changed. A definition is changed if it has been modified since one of the following events occurred:

- The file was read in (numeric argument 1 or no argument).

- The file was read in or saved (numeric argument 2).
- The definition was compiled (numeric argument 3).

Edit Changed Definitions

Command

Keystroke: META-X Edit Changed Definitions

Identifies any definitions that have been edited. This command examines all buffers for changed definitions. A definition is changed if it has been modified since one of the following events occurred:

- The file was read in (numeric argument 1 or no argument).
- The file was read in or saved (numeric argument 2).
- The definition was compiled (numeric argument 3).

When you do not specify a numeric argument, the cursor is placed at the beginning of the first changed definition.

List Changed Definitions

Command

Keystroke: META-X List Changed Definition

Lists any definitions that have been edited. This command examines all buffers for changed definitions. A definition is changed if it has been modified since one of the following events occurred:

- The file was read in (numeric argument 1 or no argument).
- The file was read in or saved (numeric argument 2).
- The definition was compiled (numeric argument 3).

Figure 3-4 provides an example of the output of this command. The command also creates a **POSSIBILITIES LISTS** buffer, which lists all the changed definitions. You can also use the **POSSIBILITIES LISTS** buffer to edit the changed definitions.

Figure 3-4

List Changed Definitions Command

```

Changed sections:

"*Buffer-1-setq-2"  COM-FIND-FILE

Type CTRL-SHIFT-P to start editing these.
```

List Buffer Changed Definitions

Command

Keystroke: META-X List Buffer Changed Definitions

Lists any definitions in the buffer that have been edited. A definition is changed if it has been modified since one of the following events occurred:

- The file was read in (numeric argument 1 or no argument).
- The file was read in or saved (numeric argument 2).
- The definition was compiled (numeric argument 3).

Figure 3-5 provides an example of the output of this command.

Figure 3-5

List Buffer Changed Definitions Command

```

Changed sections in buffer BUFFER NAME.

"*Buffer-1*-setq-2"    COM-FIND-FILE

Type CTRL-SHIFT-P to start editing these.
```

List Sections

Command

Keystroke: META-X List Sections

Lists all sections (definitions) in a specified buffer. You can press CTRL-SHIFT-F to specify a filename. Each **defun**, **defvar**, **defstruct**, and so on, is one section. You can also invoke this command from the Top-Level Editor menu.

Figure 3-6 shows a sample listing from the List Sections command.

Figure 3-6

List Sections Command

```

Sections in buffer *BUFFER-2*:

"Buffer header"      COM-GOTO-BEGINNING    COM-UP-REAL-LINE
COM-COMPILE-BUFFER  COM-GOTO-END
```

**Miscellaneous
Buffer Commands**

3.2.6 These commands allow you to perform the following operations:

- Show modifications made to the buffer.
- Treat the buffer as if it has not been modified.
- Rename a buffer.
- Sectionize a buffer.
- Make a buffer read-only or make it modifiable.
- View a buffer.

List Modifications Command

Keystroke: META-X List Modifications

Lists the lines in the buffer that have changed since the file was last saved.

Not Modified Command

Keystroke: META-~

Treats the buffer as if it has not been modified.

Rename Buffer Command

Keystroke: META-X Rename Buffer

Renames the current buffer.

Sectionize Buffer Command

Keystroke: META-X Sectionize Buffer

Reparses a buffer for definitions. This command repeats the processing normally performed only when the file is read into the buffer, that is, the processing that finds the definitions in the file so that the Edit Definition command (META-.) can work. This command is useful if you have added functions to the file.

For a detailed discussion of sectionizing, refer to paragraph 2.10.4, Lisp Code in Buffers.

Toggle Read Only Command

Keystroke: CTRL-X R

Makes the current buffer read-only, or makes it modifiable.

View Buffer Command

Keystroke: CTRL-X V

Views the contents of the specified buffer. You cannot edit the buffer. While viewing, you can press the following keystrokes to scroll:

- CTRL-V, CTRL-↓, or the space bar to scroll forward a screen of text
- META-V or CTRL-↑ to scroll backward a screen of text
- CTRL-N or ↓ to scroll forward one line
- CTRL-P or ↑ to scroll backward one line

To exit, press END.

**Print Buffer
Commands**

3.2.7 The Print Buffer commands allow you to print a buffer. In addition to the commands listed here, you can also print a buffer with the List Buffers command.

Print All Buffers Command

Keystroke: META-X Print All Buffers

Query prints all buffers on the default hard-copy device (provided that there is one). You are asked about each buffer individually.

Also refer to the Print commands.

Print Buffer Command

Keystroke: META-X Print Buffer

Prints a buffer on the default hard-copy device (provided that there is one). You are prompted in the minibuffer for the name of the buffer. The default is the current buffer. You can press CTRL-SHIFT-F to specify a filename.

Also refer to the Print commands.

Quick Print Buffer Command

Keystroke: META-SHIFT-P

Prints the current buffer on the default hard-copy device.

Also refer to the Print commands.

**Revert Buffer
Command**

3.2.8 The Revert Buffer command allows you to remove changes you made to the buffer.

Revert Buffer

Command

Keystroke: CTRL-X CTRL-R

Discards changes made to a specified buffer. You are prompted in the minibuffer for the name of the buffer. This command reads the file from disk again, deleting the buffer's former contents.

If you execute this command on a Dired buffer, the directory is read again to make a new listing.

**Select Buffer
Commands**

3.2.9 The Select Buffer commands allow you to select a buffer for editing. They allow you to switch between the buffers you are editing.

Select Buffer

Command

Keystroke: CTRL-X B

Selects the specified buffer. You are prompted in the minibuffer for the buffer name (completion available). With a numeric argument, this command allows you to create a new buffer. It also creates a new buffer if you specify the name of a nonexistent buffer. In this case, the command tells you the buffer does not exist and asks if you want to create it.

Select Default Previous Buffer

Command

Keystroke: CTRL-X META-CTRL-L

Rotates the stack of previously selected buffers. A numeric argument specifies the number of entries to rotate and sets the new default.

Select Previous Buffer

Command

Keystroke: META-CTRL-L

Selects the previously selected buffer. A numeric argument of n selects the n th previous buffer (the default argument is 2). With an argument of 1, this command rotates through the entire buffer history. A negative argument rotates through the buffer history in the opposite direction. This command uses the order of buffers that is displayed by List Buffers.

Compile and Evaluate Commands

3.3 Closely tied to the other utilities on the system, Zmacs uses the compiler and the interpreter to allow you to compile and evaluate Lisp code from within Zmacs.

When you *evaluate* Lisp code, the interpreter processes one Lisp form and executes it. Then it interprets the next Lisp form and executes it.

When you *compile* Lisp code, the compiler translates all Lisp functions into machine instructions. The code always executes whether it is compiled or interpreted. However, a defining construct that begins with an opening parenthesis in the first column followed by a **def**, such as **defun**, **defmacro**, and so on, only executes to create internal structures. (Only the top-level forms are executed.) A top-level form is a form that is not nested in another form.

The following list provides reasons for compiling a function rather than evaluating it:

- The compiled function runs much faster.
- The compiler provides better debugging information than the interpreter. The compiler prints warnings and records them in a database. You can use the database to edit the functions that received warnings. This database makes it easier to edit the source to correct the problems indicated by the error messages. It is unnecessary to write down all the messages before you begin editing.

NOTE: The compiler evaluates all top-level forms as well as some other forms, particularly when **eval-when** is used.

The following list provides reasons for evaluating rather than compiling a function:

- Evaluation is faster on a large file.
- You can execute fragments of code.
- You can create a function and have it interpreted at run time.
- Although a Lisp function runs faster if you compile it, certain debugging options are available only if you interpret the function (evaluate it). You can trace and step through a function when using evaluation. When you step through a function, you execute one Lisp form at a time. When you trace a function, you can see the order in which the Lisp forms are executed and usually the results of that execution. You cannot trace a compiled function because tracing tells you only the arguments with which the function was entered and what it returned. For more information on tracing and stepping, refer to *Explorer Tools and Utilities*.

Interpreting a function makes it actually exist in memory (that is, it is remembered by the computer). Typing a function into an editor buffer does not make it exist in memory. When you evaluate a function, the interpreter reads the definition and stores it in the package you are using.

When you compile a function, it is stored either in memory or on disk, depending on which of these methods you use to compile:

- **Compiling a file** — If you compile with the Compile File command, which uses the `compile-file` function, the file on disk is compiled, and the compiled version is put on disk with the same name and a different type, `xld`. The Compile File command does not change the memory from which you execute. If you have a buffer connected to the file and you have made changes to the buffer, you are first asked if you want to save the buffer so that your changes will be in the file on disk. To use the compiled code, you then need to *load* the `xld` file into memory by using the Load File command.
- **Compiling a buffer, region, or changed definition** — If you compile a buffer, region, or changed definition, the memory from which you execute is changed. You cannot retrieve the compiled code, and it is there only until you cold boot. The following differences exist when you compile a buffer, region, or changed definition instead of a file:
 - Incremental loading is used. Each top-level form is compiled and then loaded. This procedure allows later forms to make use of the previously compiled forms. If you use the Compile File command and then load the compiled code into memory, the benefit of using previously compiled forms is not provided.
 - You can test the buffer, region, or changed definition.
 - Compiling a region or changed definition several times until the code is right saves recompiling the whole buffer up to the point where the error occurs.

The Compile and Evaluate commands are very similar. They have the same options except that you are either evaluating or compiling. The major difference between the commands is that you cannot evaluate a file and store it in another file. Basically, you would be storing the same code. The following commands are discussed in subsequent paragraphs:

- **Evaluate Minibuffer command** — Evaluates forms in the editor.
- **Evaluate Into Buffer commands** — Evaluate forms and insert the results into your buffer.
- **Compile or Evaluate Region or Definition commands** — Compile or evaluate a region or definition.
- **Compile or Evaluate Changed Definitions commands** — Compile or evaluate only the changed definitions in a buffer.
- **Compile or Evaluate Buffer commands** — Compile or evaluate a buffer.
- **Compile File command** — Compiles a file on disk.
- **Update Xld command** — Updates an `xld` file on disk.
- **Compile and Load commands** — Compile a file on disk and load it into memory for use.

- Compiler Warning commands — Allow you to use compiler warnings.
- Disassemble commands — Enable you to obtain a more readable textual representation of the compiled code of a function.

Evaluate Minibuffer Command

3.3.1 The Evaluate Minibuffer command (META-ESCAPE) allows you to evaluate forms in the minibuffer.

Evaluate Minibuffer

Command

Keystroke: META-ESCAPE

Evaluates a form that you type into the minibuffer. The result appears in the minibuffer.

Evaluate Into Buffer Commands

3.3.2 The Evaluate Into Buffer commands allow you to evaluate a form and insert the result into your buffer.

Evaluate and Replace Into Buffer

Command

Keystroke: META-X Evaluate and Replace Into Buffer

Evaluates the symbolic expression after point and places the result into the buffer at point. The original expression is deleted, and the value that is printed out replaces it.

Also refer to paragraph 3.2.1, Capture Into Buffer Commands.

Evaluate Into Buffer

Command

Keystroke: META-X Evaluate Into Buffer

Evaluates a form from the minibuffer and inserts the result into the buffer at point. If the form returns multiple values, each value is printed into the buffer with a carriage return before each one. If you supply a numeric argument with this command, output printed by the evaluation also goes in the buffer.

Also refer to paragraph 3.2.1, Capture Into Buffer Commands.

Compile or Evaluate Region or Definition Commands

3.3.3 You can compile or evaluate a region or a definition. A Zmacs buffer that contains Lisp code is divided into sections, each containing a definition. A definition is any defining construct that begins with an opening parenthesis in the first column followed by a **def**, such as **defflavor**, **defmethod**, and **defvar**.

To compile or evaluate a region, you mark the region and then enter one of the commands in this command group.

You can use the following two procedures to compile or evaluate a definition:

- You enter one of these commands when the cursor is in the middle of a definition or immediately before or after the definition.
- You mark the definition as a region and then enter one of these commands.

The Evaluate Region, Evaluate Region Verbose, and Compile Region commands differ from Evaluate Region Hack in the way they affect a `defvar`. If you evaluate or compile a `defvar` with Evaluate Region, Evaluate Region Verbose, or Compile Region by marking the `defvar` as a region, the variable *is not* reset. This feature allows you to avoid excessive initializations that may never be used. If you use the cursor position method to evaluate or compile the `defvar`, the variable *is* reset. The Evaluate Region Hack command resets the variable in either case.

To evaluate a region or a definition, you enter one of the following commands:

Evaluate Region	Command
------------------------	----------------

Keystroke: CTRL-SHIFT-E

Evaluates the current region or definition. The result is returned in the minibuffer. If there is a region, it is evaluated. Otherwise, the current or next definition is evaluated.

If you evaluate a `defvar`, this command works differently depending on how you evaluate it:

- If you evaluate the `defvar` by marking it as a region, the variable *is not* reset.
- If you evaluate the `defvar` by the position of the cursor, the variable *is* reset even if it is already bound.

Evaluate Region Hack	Command
-----------------------------	----------------

Keystroke: META-CTRL-SHIFT-E

Evaluates the current region or definition. If there is a region, it is evaluated. Otherwise, the current or next definition is evaluated.

If you evaluate a `defvar` by either marking it as a region or by the position of the cursor, the variable is reset even if it is already bound.

Evaluate Region Verbose	Command
--------------------------------	----------------

Keystroke: META-SHIFT-E

Evaluates the current region or definition. This command differs from the Evaluate Region command in that the result is returned in the typeout window rather than in the minibuffer. If there is a region, it is evaluated. Otherwise, the current or next definition is evaluated.

If you evaluate a **defvar**, this command works differently depending on how you evaluate it:

- If you evaluate the **defvar** by marking it as a region, the variable *is not* reset.
- If you evaluate the **defvar** by the position of the cursor, the variable *is* reset even if it is already bound.

To compile a region or definition, you enter the following command:

Compile Region

Command

Keystroke: CTRL-SHIFT-C

Compiles the current region or definition. If there is a region, it is compiled. Otherwise, the current or next definition is compiled.

If you compile a **defvar**, this command works differently depending on how you compile it:

- If you compile the **defvar** by marking it as a region, the variable *is not* reset.
- If you compile the **defvar** by the position of the cursor, the variable *is* reset even if it is already bound.

Compile or Evaluate Changed Definition Commands

3.3.4 These commands compile or evaluate only the definitions of any buffers that you have changed since you last compiled or evaluated. These commands keep you from having to compile or evaluate entire buffers. When you change a portion of a buffer, you can mark it as a region and compile or evaluate it. However, if you make changes throughout the buffer without compiling or evaluating or if you do not remember what you changed, you can use these commands rather than compiling or evaluating the entire buffer.

When Zmacs reads into a buffer a file that contains Lisp code, it divides the definitions in the buffer into sections. The only buffers that have sections are those containing Lisp code. Other buffers do not have sections. A *section* can contain any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defvar**, **defflavor**, and **defun**. The defining construct can be user-defined. Each definition is a section.

There is also a section at the top of the buffer called the *buffer header*, which essentially contains the attribute list.

For a detailed discussion of this section scheme, refer to paragraph 2.10.4, *Lisp Code in Buffers*.

The Compile and Evaluate Changed Definitions commands compile or evaluate those sections (definitions) that have a section tick greater than their compile or evaluate tick. The *tick* is a counter that increments each time any operation is performed. For example, the compile tick records the current tick value each time you compile; the section tick records the current tick value each time the section is updated. Thus, a comparison of the compile and section ticks tells the editor if the current definition in a section is the compiled value of the function defined there.

Compile Buffer Changed Definitions Command

Keystroke: META-X Compile Buffer Changed Definitions

Compiles any definitions in this buffer that have been edited. With a numeric argument, this command asks about each definition individually.

Compile Changed Definitions Command

Keystroke: META-X Compile Changed Definitions

Compiles any definitions that have been edited in any of the buffers. With a numeric argument, this command asks about each definition individually.

Evaluate Buffer Changed Definitions Command

Keystroke: META-X Evaluate Buffer Changed Definitions

Evaluates any definitions in this buffer that have been edited. With a numeric argument, this command asks about each definition individually.

Evaluate Changed Definitions Command

Keystroke: META-X Evaluate Changed Definitions

Evaluates any definitions that have been edited in any of the buffers. With a numeric argument, this command asks about each definition individually.

Compile or Evaluate Buffer Commands

3.3.5 You can either compile or evaluate a buffer. To evaluate an entire buffer, you enter one of the following commands:

Evaluate and Exit Command

Keystroke: META-CTRL-Z

Evaluates the entire buffer. If the buffer evaluates with no errors, this command leaves the editor and returns to the program you were using before you entered the editor, such as the Lisp Listener.

Evaluate Buffer Command

Keystroke: META-X Evaluate Buffer

Evaluates the entire buffer.

Compile and Exit Command

Keystroke: META-Z

Compiles the entire buffer. Then, this command leaves the editor and returns to the program you were using before you entered the editor, such as the Lisp Listener.

Compile Buffer Command

Keystroke: META-X Compile Buffer

Compiles the entire buffer but does not exit.

**Compile File
Command**

3.3.6 The Compile File command enables you to compile a file on disk.

Compile File Command

Keystroke: META-X Compile File

The Compile File command, which uses the **compile-file** function, compiles a file on disk and puts the compiled version on disk with the same filename and a different type, xld. This command does not change the memory from which you execute. If you have a buffer connected to the file and you have made changes to the buffer, you are first asked if you want to save the buffer so that your changes will be in the file on disk. To use the compiled code, you then need to *load* the xld file into memory, which is a separate step. Refer to paragraph 3.3.8, Compile and Load Commands.

If a file uses a function or a global variable, warnings are given when you try to compile the file, unless the function is already in the system. If it is not in the system, you can bring the file containing the function into the editor by using the Find File command (CTRL-X CTRL-F) and then evaluate or compile it. You can also load the file containing the function. You can load an xld or a Lisp file.

If you use a macro that does not currently exist in the system, the file that you are trying to compile does not compile correctly. You must load macros before you can compile files that use them. The same is true for flavors, methods, **defstruct**, and various other items.

Update Xld Command 3.3.7 The Update Xld command allows you to compile the code in your buffer and write it back to disk as an xld file.

Update Xld Command

Keystroke: CTRL-X META-CTRL-SHIFT-C

Updates the xld file of the file that you are visiting. This command uses the function definitions present in the environment, offering to compile them if they have changed. Note that `declare` and `eval-when (compile)` are ignored.

Compile and Load Commands 3.3.8 The Compile and Load commands allow you to compile a file, if the xld file has an earlier date than the file, and load the file so that you can use the most up-to-date versions of the functions it contains. Loading means to bring the contents on *disk* into *memory* so that the functions and so on can be called. This operation differs from the Compile File command, which compiles the file on *disk* and stores the compiled version on *disk* in an xld file without affecting what is currently in memory.

Compile and Load File Command

Keystroke: META-X Compile and Load File

Loads a file after compiling it if the xld file has an earlier date than the file.

This command compiles the file on disk and stores the compiled version on disk. Then it loads the compiled version into memory. It is equivalent to executing the Compile File command followed by the Load File command.

Load File Command

Keystroke: META-X Load File

Loads a file. With a numeric argument, this command compiles the file if the xld file has an earlier date than the file.

You can load either an xld or a Lisp file. If you do not specify a type, the command looks for xld first and then Lisp. If the command loads Lisp, it is interpreted. If the command loads xld, it is compiled.

Loading a Lisp file with this command is equivalent to executing the Find File command followed by the Evaluate Buffer command.

**Compiler Warning
Commands**

3.3.9 When the compiler prints warnings and error messages, it also records them in a database, organized by file and by function within the file. Old warnings for previous compilations of the same function are discarded; the database contains only warnings that are still applicable. You can use this database to visit the functions that receive warnings. You can also save the database and restore it later.

You can use one of three editor commands to begin visiting the sites of the recorded warnings. These commands differ only in how they decide which files to look through:

- Edit Warnings (META-X Edit Warnings)
- Edit File Warnings (META-X Edit File Warnings)
- Edit System Warnings (META-X Edit System Warnings)

For information on systems, refer to the *Explorer Lisp Reference*.

While the warning sites are being edited, the warnings themselves appear in a small window at the top of the editor buffer window, and the code appears in a large window that occupies the rest of the editor buffer window.

As soon as you have finished specifying the file(s) or system to process, the editor visits the code for the first warning. From then on, you can move to the next warning by using the Edit Next Warning command (CTRL-SHIFT-W). To move to the previous warning, you use the Edit Previous Warning command (META-SHIFT-W). You can also switch to the warnings window with the Other Window command (CTRL-X O) or with the mouse; then you can move around in that buffer. When you use Edit Next Warning (CTRL-SHIFT-W) and there are no more warnings after the cursor, you return to one-window mode.

You can also insert the text of the warnings into any editor buffer by using one of the following commands:

- Insert File Warnings (META-X Insert File Warnings)
- Insert Warnings (META-X Insert Warnings)

You can also dump the database that contains the warnings into a file and reload it later. Then you can use the Edit Warnings command again in the later session. For information on dumping the warnings and loading the file later, refer to the *Explorer Lisp Reference*.

NOTE: If the code you are compiling causes the compiler to terminate with an error, the location of the faulty code is stored in a register named . (period). You can use the Jump to Saved Position command (CTRL-X J) to go to the location saved in the register named . (period).

Edit Warnings

Command

Keystroke: META-X Edit Warnings

Edits warnings from compilations and similar operations. First, you are asked which files' warnings you want to edit. Then, a list of all warnings for these files is placed in the buffer *WARNINGS*. You move through that buffer in a small top window, with the corresponding code appearing in the bottom window.

When you finish editing the last warning, press CTRL-SHIFT-W (the Edit Next Warning command) to remove the small warnings window and return to one-window mode. To return to one-window mode before you finish editing the warnings, press CTRL-X 1. (This command is the One Window command.)

Edit Compiler Warnings

Command

Keystroke: META-X Edit Compiler Warnings

This command is a synonym for the Edit Warnings command.

Edit File Warnings

Command

Keystroke: META-X Edit File Warnings

Edits warnings for a particular file. This command is similar to Edit Warnings except that you specify one filename and only the warnings for that file are processed.

When you finish editing the last warning, press CTRL-SHIFT-W (the Edit Next Warning command) to remove the small warnings window and return to one-window mode. To return to one-window mode before you finish editing the warnings, press CTRL-X 1. (This command is the One Window command.)

Edit System Warnings

Command

Keystroke: META-X Edit System Warnings

Edits warnings for the files in a specified system. This command is similar to Edit Warnings except that you specify the name of a system and the warnings for the source files of that system are processed.

When you finish editing the last warning, press CTRL-SHIFT-W (the Edit Next Warning command) to remove the small warnings window and return to one-window mode. To return to one-window mode before you finish editing the warnings, press CTRL-X 1. (This command is the One Window command.)

Edit Next Warning Command

Keystroke: CTRL-SHIFT-W

Edits the next function that has a warning. Once you have started editing a list of warnings with `Edit Warnings` or `Edit File Warnings`, this command moves to the next function with warnings or error messages.

Edit Previous Warning Command

Keystroke: META-SHIFT-W

Edits the previous warning's function. Once you have started editing a list of warnings with `Edit Warnings` or `Edit File Warnings`, this command returns to the previous warning.

Go to Warning Command

Keystroke: CTRL-/

If point is at a warning in the small top window, this command goes to the text in the large window that the warning describes.

Insert File Warnings Command

Keystroke: META-X Insert File Warnings

Inserts at point the warnings about a file. You are prompted in the minibuffer for the name of the file. A numeric argument specifies the buffer of warnings for compile operations not associated with files (such as compiling from the Lisp Listener by calling `compile`). This command leaves the mark after the inserted text, but the region is not turned on.

Insert Warnings Command

Keystroke: META-X Insert Warnings

Inserts at point all the warnings about all files. This command leaves the mark after the inserted text, but the region is not turned on.

**Disassemble
Commands**

3.3.10 The Disassemble commands read the compiled code of a function and provide a readable textual representation of the code. *Disassemble* means reverse assemble. In other words, instead of producing executable code from assembly language, it produces a textual representation of the executable code. For more information, refer to the *Explorer Lisp Reference*.

Disassemble

Command

Keystroke: META-X Disassemble

Disassembles the specified function. You are prompted in the minibuffer for the name of the function.

Quick Disassemble

Command

Keystroke: META-X Quick Disassemble

Disassembles the function to the left of the keyboard cursor.

Cursor Movement Commands

3.4 Zmacs provides a wide range of Cursor Movement commands. The five groups of Cursor Movement commands are as follows:

- **General Cursor Movement commands** — Allow you to move up or down a real line. (A real line is a line that is terminated by a carriage return.) They also allow you to move to the beginning or end of a line, to the top or bottom of the buffer, and to the top or bottom of the screen.
- **Lisp Cursor Movement commands** — Allow you to move backward or forward a symbolic expression, list, or definition in Common Lisp or Zetalisp mode. They also allow you to move up or down a comment line.
- **Text Cursor Movement commands** — Allow you to move backward or forward a character, word, sentence, or paragraph in Text mode.
- **Saving Cursor Locations (Point) commands** — Allow you to store cursor locations and return to them later.
- **Various Quantities command** — Allows you to move the cursor by various quantities. A *quantity* is a character, word, sentence, symbolic expression, and so on. (This command also allows you to perform operations such as delete, transpose, and copy.)

In addition to these commands, you can use the mouse to move the keyboard cursor. You position the mouse cursor where you want the keyboard cursor and click left.

Remember that you can add a numeric argument to these operations to perform more than one item at a time. For example, you can move 20 characters forward instead of one by typing CTRL-2 CTRL-0 CTRL-F.

General Cursor Movement Commands

3.4.1 The following paragraphs describe the General Cursor Movement commands:

Backward Character Command

Keystroke: CTRL-B or ←

Moves point one or more characters backward. With a numeric argument, this command moves point the specified number of characters backward.

Beginning of Line Command

Keystroke: CTRL-A or SUPER-←

Moves to the beginning of the line. With a numeric argument, this command also moves forward by a number of lines one less than the argument.

Down Real Line Command

Keystroke: CTRL-N or ↓

Moves down vertically to the next real line.

End of Line Command

Keystroke: CTRL-E or SUPER-→

Moves to the end of the line. With a numeric argument, this command also moves forward by a number of lines one less than the argument.

Forward Character Command

Keystroke: CTRL-F or →

Moves point one or more characters forward. With a numeric argument, this command moves point the specified number of characters forward.

Goto Beginning Command

Keystroke: META-< or HYPER-↑

Goes to the beginning of the buffer. With an argument from 0 to 10, this command goes forward that many tenths of the length of the buffer starting from the beginning.

Also refer to paragraph 3.17, Scroll Commands.

Goto End Command

Keystroke: META-> or HYPER-↓

Goes to the end of the buffer. With an argument from 0 to 10, this command goes backward that many tenths of the length of the buffer starting from the end.

Also refer to paragraph 3.17, Scroll Commands.

Move to Bottom of Screen Command

Keystroke: SUPER-↓

Moves vertically to the bottom of the screen. This command moves to the last line on the screen and either stays in the same column, if possible, or moves to the rightmost character in the line.

Move to Screen Edge Command

Keystroke: META-R

Positions the keyboard cursor on the left margin of the screen as determined by the Zmacs user variable `Center Fraction`. The default is the middle of the screen. A numeric argument specifies the screen line where you want to go. Negative arguments count upward from the bottom of the screen.

Move to Top of Screen Command

Keystroke: SUPER-↑

Moves vertically to the top of the screen. This command moves to the first line on the screen and either stays in the same column, if possible, or moves to the rightmost character in the line.

Up Real Line Command

Keystroke: CTRL-P or ↑

Moves up vertically to the previous real line.

Lisp Cursor Movement Commands

3.4.2 Many of the Lisp Cursor Movement command explanations use the terminology shown in the following table to describe their operation:

Term	Description
Forward list	Moves forward to the next element at the current level.
Backward list	Moves backward to the next element at the current level.
Up list	Moves up into an element.
Down list	Moves down into an element.

Suppose you perform the forward list and backward list operations on the following code with the cursor on the `e`:

```
(a b (d (g h i) e f) c)
```

The following shows where the cursor moves:

Forward list: (a b (d (g h i) e f) **█** c)

Backward list: (a b (d **█** g h i) e f) c)

Suppose you perform the up list and down list operations on the following code with the cursor on the d:

```
(a b (d (g h i) e f) c)
```

The following shows where the cursor moves:

```
Up list: (a b (d (g h i) e f) c)
```

```
Down list: (a b (d (g h i) e f) c)
```

Back to Indentation

Command

Keystroke: META-M

Moves to the first nonblank character in the current line. If there is a fill prefix, this command moves to the first nonblank character after the fill prefix (even if the fill prefix is not blank).

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Backward Down List

Command

Keystroke: META-X Backward Down List

Moves down one or more levels of list structure, backward. In other words, this command moves backward over the elements of the current list until it finds one that is a list. Then it positions the cursor at the end of this inner list.

For example, suppose you execute this command on the following code with the cursor on the e:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b (d (g h i) e f) c)
```

Backward List

Command

Keystroke: META-CTRL-P

Moves backward one or more lists. For example, suppose you execute this command on the following code with the cursor on the e:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b (d (g h i) e f) c)
```

Backward Sexp Command

Keystroke: META-CTRL-B or META-CTRL-←

Moves one or more symbolic expressions backward.

Backward Sexp No Up Command

Keystroke: META-X Backward Sexp No Up

Moves backward one or more symbolic expressions, but never over an unbalanced opening parenthesis. This command is useful in keyboard macros, for example.

Backward Up List Command

Keystroke: META-CTRL-(

Moves up one level of list structure, backward. Also, if called inside a string, this command moves back up out of that string.

Suppose you execute this command on the following code with the cursor on the e:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b █ d (g h i) e f) c)
```

Beginning of Definition Command

Keystroke: META-CTRL-[or META-CTRL-↑

Moves to the beginning of the current definition (such as **defun**, **defflower**, and so on).

Down Comment Line Command

Keystroke: META-N

Moves to the comment position in the next line and inserts a semicolon (;) if one is not already there. This command is equivalent to the Down Real Line command (CTRL-N) followed by the Indent for Comment command (CTRL-;), except that any blank comment on the current line is deleted before the keyboard cursor is moved.

Down Indented Line Command

Keystroke: META-X Down Indented Line

Moves to the first nonblank character on the next line.

Down List Command

Keystroke: META-CTRL-D

Moves down one or more levels of list structure.

Suppose you execute this command on the following code with the cursor on the d:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b (d (g h i) e f) c)
```

End of Definition Command

Keystroke: META-CTRL-] or META-CTRL-↓

Moves to the end of the current definition (such as **defun**, **defflower**, and so on).

Forward List Command

Keystroke: META-CTRL-N

Moves forward one or more lists. With a numeric argument, this command moves the specified number of lists forward.

Suppose you execute this command on the following code with the cursor on the d:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b (d (g h i) e f) c)
```

Forward Sexp Command

Keystroke: META-CTRL-F or META-CTRL-->

Moves one or more symbolic expressions forward.

Forward Sexp No Up Command

Keystroke: META-X Forward Sexp No Up

Moves forward one or more symbolic expressions, but never over an unbalanced closing parenthesis. This command is useful in keyboard macros, for example.

Forward Up List Command

Keystroke: META-CTRL-)

Moves up one level of list structure, forward. Also, if called inside a string, this command moves up out of that string.

Suppose you execute this command on the following code with the cursor on the d:

```
(a b (d (g h i) e f) c)
```

The following line shows where the cursor moves:

```
(a b (d (g h i) e f) c)
```

Move Over) Command

Keystroke: META-)

Moves over the next closing parenthesis, inserting return characters and updating indentation. Any indentation before the closing parenthesis is deleted. Lisp-style indentation is inserted after the closing parenthesis.

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Up Comment Line Command

Keystroke: META-P

Moves to the comment position in the previous line and inserts a semicolon (;) if one is not already there. This command is equivalent to the Up Real Line command (CTRL-P) followed by the Indent for Comment command (CTRL-;), except that any blank comment on the current line is deleted before the keyboard cursor is moved.

Up Indented Line Command

Keystroke: META-X Up Indented Line

Moves to the first nonblank character on the previous line.

**Text Cursor
Movement
Commands**

3.4.3 The following paragraphs describe the Text Cursor Movement commands:

Backward Paragraph

Command

Keystroke: META-[

Moves to the start of this (or the last) paragraph. Paragraphs are delimited by blank lines or by lines that start with a delimiter in the Zmacs user variable Paragraph Delimiter List or in the user variable Page Delimiter List. If there is a fill prefix, any line that does not start with the fill prefix starts a paragraph. If a line starts with a character that is in the user variable Text Justifier Escape List and that is also in the user variable Paragraph Delimiter List, the line counts as a blank line because it separates paragraphs and is not part of them. You can see and change the values of these variables by executing the Variable Apropos command (HELP A).

Backward Sentence

Command

Keystroke: META-A or META-↑

Moves to the beginning of the sentence. A sentence is ended by a period (.), question mark (?), or exclamation point (!) and followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote ("), single apostrophe ('), right parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.

Backward Word

Command

Keystroke: META-B or META-←

Moves one or more words backward.

Forward Paragraph

Command

Keystroke: META-]

Moves to the start of the next paragraph. Paragraphs are delimited by blank lines or by lines that start with a delimiter in the Zmacs user variable Paragraph Delimiter List or in the user variable Page Delimiter List. If there is a fill prefix, any line that does not start with the fill prefix starts a paragraph. If a line starts with a character that is in the user variable Text Justifier Escape List and that is also in the user variable Paragraph Delimiter List, the line counts as a blank line because it separates paragraphs and is not part of them. You can see and change the values of these variables by executing the Variable Apropos command (HELP A).

Forward Sentence

Command

Keystroke: META-E or META-↓

Moves to the end of this sentence. If the cursor is already at the end of a sentence, it moves to the end of the next sentence. A sentence is ended by a period (.), question mark (?), or exclamation point (!) and followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote (”), single apostrophe (’), right parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.

Forward Word

Command

Keystroke: META-F or META-→

Moves one or more words forward.

**Saving Cursor
Locations (Point)
Commands**

3.4.4 The Saving Cursor Locations (Point) commands allow you to store cursor locations (points) and return to them later. The cursor locations can be in different buffers, allowing you to jump between buffers.

Many of these commands store points on the *point PDL*, which is a list. The point PDL usually stores up to eight points. New entries are put on the top of the point PDL. When the point PDL is full, entries on the bottom are deleted as new entries are added.

Other commands store points in *registers*. A register can store only one point at a time, but you do not have to worry about an entry being deleted. Besides storing points, some of these commands manipulate mark. The following list summarizes the operations you can perform with the Saving Cursor Locations (Point) commands:

- Push a point onto the point PDL
- Exchange point with the top entry on the point PDL
- Rotate through the point PDL
- Show the point PDL
- Save the current point in a register
- Restore a saved position from a register
- Show the positions saved in registers
- Exchange point and mark
- Set the mark to point

Section 2 describes how to use the point PDL and registers as well as how to mark.

Jump to Saved Position Command

Keystroke: CTRL-X J

Restores a saved position from a register. You are prompted in the minibuffer for the register name, a character with no attribute bits. (The term *attribute bits* means a key used in combination with SUPER, HYPER, META, or CTRL.)

Also refer to paragraph 3.6.4, Register Commands.

Move to Default Previous Point Command

Keystroke: CTRL-X META-CTRL-space bar

Returns to the second most recent entry on the point PDL when first pressed. Pressing CTRL-X META-CTRL-space bar again returns to the most recent entry on the point PDL. Pressing a third time returns to the original point.

Move to Previous Point Command

Keystroke: META-CTRL-space bar

Returns to the previous point on the point PDL. This command exchanges the current location with the previous one. A numeric argument specifies the number of entries on the top of the point PDL to rotate through (the default numeric argument is 2). An argument of 1 rotates forward through the whole point PDL, and a negative argument rotates in the other direction.

For example, with a numeric argument of 4, the current point is pushed on the point PDL and the third most recent point (now four from the top because of the push) is removed from the point PDL and becomes the value of point.

Push Pop Point Explicit Command

Keystroke: META-space bar

Pushes or pops point onto the point PDL. With no argument, this command pushes point onto the point PDL. With an argument, this command exchanges point with the *n*th position on the stack.

Save Position Command

Keystroke: CTRL-X S

Saves the current point in a register. You are prompted in the minibuffer for the register name, a character with no attribute bits. (The term *attribute bits* means a key used in combination with SUPER, HYPER, META, or CTRL.)

Set Pop Mark

Command

Keystroke: CTRL-space bar

Sets or pops the mark in the following ways:

- Without a CTRL-U argument, this command sets mark at point and pushes point onto the point PDL.
- With one CTRL-U, this command moves point down the point PDL. (The top entry on the point PDL becomes point.) Each time you press CTRL-U CTRL-space bar, point moves farther down the point PDL, leaving mark at the original position for easy return with the Swap Point and Mark command (CTRL-X CTRL-X).
- With two CTRL-Us, this command pops the point PDL and discards it. In other words, the top entry on the point PDL becomes point, and this entry is discarded from the point PDL.

Also refer to paragraph 3.6.3, Mark Commands.

Show Point PDL

Command

Keystroke: META-STATUS

Shows the contents of the point PDL. The point PDL lists the last eight locations of point. The display is mouse-sensitive except for the first line. You click left on the line to which you want to go.

The first line tells you the buffer from which point came. If the buffer contains Lisp code, the first line also tells the section from which point came. For example, it might indicate that the three lines are from a section called **com-find-file** in the buffer called ***BUFFER-3***. This information is helpful if you have a few lines of code from the middle of a function and you want to know from what function it came.

With a numeric argument, this command shows n lines for each entry on the point PDL, where n equals the numeric argument. The default argument is 3.

Figure 3-7 shows an example point PDL listing.

Figure 3-7 Show Point PDL Command

This is a list of locations to which you may jump.
 META-STATUS gives this display; arg. controls the number of lines shown in one entry.

Other commands that manipulate the point pdl are:

META-SPACE with no numeric argument adds your current position to the list.

With a numeric argument n, exchange the current and nth points

CTRL-META-SPACE rotates through the current & last 2 locations. Args. change 2.

The MOUSE is like CTRL-META-SPACE with an argument but rotates the other direction.

- 1 from Buffer header in GETTYSBURG-INSERT.TEXT#> TI-EXAMPLE.ZMACS; L10:
 freedom and that government of the people, by the people, for the people,
 shall not perish from the earth.
- 2 from Buffer header in GETTYBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; L10:
 we have come to
 dedicate a portion of that field as a final resting place for those who here
 gave their lives that the nation might live. The world will little note,
- 3 from COM-FIND-FILE in ZMACS.LISP#> ZWEI; L10:
 (DEFCOM COM-FIND-FILE "Visits a file in its own buffer.
 Reads in a filename from the minibuffer. If the file is already
 in a buffer, selects that buffer. Otherwise creates a buffer
- 4 from Buffer header in GETTYSBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; L10:
 We have come to
 dedicate a portion of that field as a final resting place for those who here
 gave their lives that the nation might live. The world will little note,

Show Saved Positions

Command

Keystroke: META-X Show Saved Positions

Gives a mouse-sensitive display of positions saved in registers with the Save Position command (CTRL-X S). This command pushes the current point on the point PDL for easy return with various commands.

Also refer to paragraph 3.6.4, Register Commands.

Swap Point and Mark

Command

Keystroke: CTRL-X CTRL-X

Exchanges point and mark.

Also refer to paragraph 3.6.3, Mark Commands.

**Various Quantities
Command**

3.4.5 The Various Quantities command allows you to move the cursor by quantities. A *quantity* is a character, word, sentence, paragraph, line, atom, page, and so on. This command also allows you to perform operations such as deleting, transposing, and copying various quantities.

PAGE characters are produced by pressing CTRL-Q CLEAR-SCREEN. A *page* is delimited by the PAGE character.

The Various Quantities command can perform either of the following operations:

- Insert characters with attribute bits or nonletters. The term *attribute bits* means a key used in combination with SUPER, HYPER, META, or CTRL.
- Manipulate various quantities.

This command uses the key sequence CTRL-Q as a prefix for both inserting and manipulating quantities.

- To insert characters, you press CTRL-Q and then the character. For example, you can press CTRL-Q CLEAR SCREEN to insert a PAGE character into your buffer.
- To manipulate quantities, you press CTRL-Q followed by another letter for the type of operation and then finally a letter for the type of quantity. Some of the operations you can perform are forward, backward, delete, copy, and transpose.

Various Quantities

Command

Keystroke: CTRL-Q

Given characters with attribute bits or nonletters, this command inserts them. A question mark (?) displays documentation about this command.


Given octal digits, this command inserts the character with the specified character code. If you specify less than three octal digits, you should insert a space after the last digit. If given a numeric argument, this command inserts the octal character the specified number of times.

Otherwise, this command manipulates various quantities:

- The first character following CTRL-Q invokes one of the following operations:

B	backward	S	save
C	copy	T	transpose
D	delete	U	uppercase
F	forward	Z	reverse
L	lowercase	@	mark region
R	rubout		

- The second character following CTRL-Q invokes one of the following quantity types:

A	atom	S	sentence
C	character	W	word
D	definition	(list
H	buffer)	list
L	line	-	symbolic expression
P	paragraph		page

You can also use numeric arguments with this command.

Customization Commands

3.5 The Customization commands allow you to change how Zmacs operates and tailor it to suit your individual needs. Zmacs contains the following commands to help you easily change certain aspects of its workings:

- Install Command on Key commands — Allow you to install a specified command or function on a specified key.
- Keyboard Macro commands — Allow you to store a sequence of commands for use as a group.
- Variable commands — Help you find and change Zmacs user variables. These variables are provided to allow you to customize Zmacs.
- Word Abbreviation commands — Allow you to use word abbreviations when you enter text into your buffer. Zmacs automatically expands the abbreviation to its definition.

Install Command on Key Commands

3.5.1 The Install Command on Key commands allow you to install a command or a function on a key.

Set Key

Command

Keystroke: META-X Set Key

Installs a specified command on a specified key. You are prompted in the minibuffer first for the name of the command to install and then for a key. If the key is currently assigned a command prefix (such as CTRL-X), you are prompted for another character so that you can redefine CTRL-X commands. However, with a numeric argument, the Set Key command assumes you want to redefine CTRL-X itself and does not ask for another character.

Install Command

Command

Keystroke: META-X Install Command

Installs a specified function on a specified key. The function must return a numeric value that represents the degree to which the window must be redisplayed. Online documentation for the function `zwei:must-redisplay` describes the symbols that contain appropriate values. You are prompted in the minibuffer first for the name of the function and then for a key. If the key is currently assigned a command prefix (such as CTRL-X), you are prompted for another character so that you can redefine CTRL-X commands. However, with a numeric argument, the Install Command assumes you want to redefine CTRL-X itself and does not ask for another character.

If you want to write your own command, you should use `zwei:defcom`. You can use the Edit Definition command (META-.) to find `zwei:defcom` and also to find examples of other commands in the editor to see how `zwei:defcom` is used.

**Keyboard Macro
Commands**

3.5.2 The Keyboard Macro commands allow you to define keyboard macros. A *keyboard macro* is a sequence of keystrokes that are called as one command. Using Keyboard Macro commands, you can press several keys that perform various commands and make these keys a repeatable sequence by defining one keystroke to call that sequence of keys.

You can do the following with these commands:

- Define a keyboard macro.
- Test the macro the number of times you want.
- Give the macro a name and store it.
- Install the macro on a key or a mouse click.

You can also nest macros within macros. You can call a macro to aid in the definition of another macro. While defining one macro, you can define another by reentering the Start Kbd Macro command (CTRL-X (). The mode line then indicates a level 2 macro definition.

Suppose you want to define a keyboard macro called Comment Line and Return that comments out a line and moves to the next line:

1. Press CTRL-X (to start the keyboard macro definition.

The mode line tells you that you are at macro level 1, meaning that you are defining a macro.

2. Press the keys CTRL-A ; CTRL-N.

This sequence of keys comments out the line and moves to the next line.

3. Press CTRL-X) to end the definition of the keyboard macro.

The mode line removes the macro notification.

4. Press CTRL-X E to test the macro to see if it does what you intended.

5. Press META-X and type Name Last Kbd Macro to give the macro you just defined a permanent name.

6. Enter the name `comment-Line-and-Return`. The name should not contain spaces.

7. Press META-CTRL-TERM C and type `comment-Line-and-Return` to try out your new command.

8. Press META-X and type `Install Macro` to install the macro on a key.

9. Press RETURN to indicate that you are installing the macro you just defined.

- 10 Press a key sequence such as the SUPER-META-; sequence.
11. Press SUPER-META-; to execute the macro.
12. Press CTRL-5 SUPER-META-; to execute the macro five times.

NOTE: The keyboard macro lasts only for your edit session unless you put it in your login init file. Use the commands Write Kbd Macro and Load Kbd Macro, described later in this numbered paragraph, to save and restore your keyboard macros. Section 4 describes login init files.

In addition to the commands discussed later in this paragraph, pressing META-CTRL-TERM HELP displays a list of other keystrokes that you can use for defining keyboard macros. The following table shows this list:

Key	Description
P	Pushes a level of macro; that is, this command allows you to define one macro while defining another macro.
R	Ends the definition and repeats the number of times specified by the numeric argument. For example, you can press CTRL-5 R to execute the macro 5 times.
C	Allows you to execute a macro by specifying its name.
S	Terminates the definition of a macro.
U	Allows type-in now only; that is, the type-in is executed, and it is not part of the macro.
T	Allows type-in in expansion also. The type-in is put on the screen and overrides the D command; that is, it is executed and recorded in the macro. (Press META-CTRL-TERM R to terminate the type-in.)
M	Defines a named macro.
D	Defines a named macro but does not execute. Normally, when you type a macro, it is executed.
A	Stores an increasing character string. You supply a character, then a number. The number of the character and the number are added together. For example, supplying L and 1 produces M.
Space bar	Creates an interactive keyboard macro. This command is the same as the Kbd Macro Query command (CTRL-X Q).
END	Cancels META-CTRL-TERM prefix and refreshes screen.

Start Kbd Macro Command

Keystroke: CTRL-X (

Begins defining a keyboard macro. With a numeric argument, this command appends to the previous keyboard macro.

End Kbd Macro Command

Keystroke: CTRL-X)

Terminates the definition of a keyboard macro.

Call Last Kbd Macro Command

Keystroke: CTRL-X E

Repeats the last keyboard macro. A numeric argument makes the macro repeat the specified number of times. However, a numeric argument of 0 makes the macro repeat infinitely. This option is useful if a user query is included in the macro. The Kbd Macro Query command (CTRL-X Q) allows you to define interactive keyboard macros.

Name Last Kbd Macro Command

Keystroke: META-X Name Last Kbd Macro

Makes the last temporary macro permanent. You are prompted in the minibuffer for the new name of the macro. The name should not contain spaces.

Install Macro Command

Keystroke: META-X Install Macro

Installs a specified user macro on a specified key. The macro should be a *permanent* macro (that is, one with a name). You are prompted in the minibuffer for the name of the macro. (The default is the last macro defined.) Then you are prompted for the keystroke on which to install it. If the key is currently holding a command prefix (such as CTRL-X), you are asked for another character so that you can redefine CTRL-X commands. However, with a numeric argument, this command assumes that you want to redefine CTRL-X itself and does not ask for another character.

Install Mouse Macro Command

Keystroke: META-X Install Mouse Macro

Installs a specified user macro on a specified mouse button. The macro should be a *permanent* macro (that is, one with a name). You are prompted in the minibuffer for the name of the macro and then the mouse button on

which to install it. (You can click the button more than once to specify a double click and so on.)

Later, when you click the mouse button to execute the macro, it moves the keyboard cursor to the mouse cursor and executes the command.

View Kbd Macro Command

Keystroke: META-X View Kbd Macro

Displays the specified keyboard macro. The macro should be a *permanent* macro (that is, one with a name). You are prompted in the minibuffer for the name of the macro.

Pressing only RETURN displays the last macro defined, even if temporary. That is, the macro either has to be permanent or the last one defined. No others are accessible.

Deinstall Macro Command

Keystroke: META-X Deinstall Macro

Deinstalls a keyboard macro from a specified key or mouse click.

Kbd Macro Query Command

Keystroke: CTRL-X Q

Creates an interactive keyboard macro. This command causes a pause when encountered in a keyboard macro so that the user can interact with the macro. This feature allows customization of the macro action at this point or termination of the macro. This command is useful (possibly required) in a macro that repeats infinitely (argument of 0).

Write Kbd Macro Command

Keystroke: META-X Write Kbd Macro

Writes the keystrokes for a named macro to a file that you specify.

Load Kbd Macro Command

Keystroke: META-X Load Kbd Macro

Loads your macro file (see the previous command, Write Kbd Macro) into your environment. If you want to load your macros from your login init file, use the following function:

```
(zwei:load-kbd-macros "your-kbd-macros.lisp")
```

Variable Commands 3.5.3 Zmacs contains user variables that are provided to allow you to customize Zmacs. You can change these variables to the values you want. For example, the Region Marking Mode variable controls the way a marked region appears. You can change the value of that variable from underlining to reverse video.

NOTE: The Variable Apropos command (HELP V) and the List Variables command (META-X List Variables) list user variables with syntax such as Page Delimiter List. The actual variable name is `zwei:*page-delimiter-list*`.

The Variable commands provide the following options:

Command	Brief Description
List Variables	Lists all user variables and their values
Variable Apropos	Helps find a user variable
Set Variable	Sets a user variable
Describe Variable	Provides documentation about a user variable
List Local Variables	Lists user variables local to the buffer
Make Local Variable	Makes a user variable local to the buffer
Kill Local Variable	Changes a user variable that is local to the buffer into a global variable

A *local* user variable applies only to the buffer where it is set. Otherwise, these user variables apply globally to all buffers.

List Variables Command

Keystroke: META-X List Variables

Lists all Zmacs user variables and their values. With an argument, this command provides documentation as well.

Variable Apropos Command

Keystroke: HELP V

Finds all the Zmacs user variables whose names contain a specified string. Zmacs provides *user variables* to allow you to customize many aspects of its workings.

The display given by this command is mouse-sensitive to allow easy modification of variable values. When you click left on any variable listed, its value

appears for editing in the minibuffer. Once the new value is in the minibuffer, press the END key to change the variable.

You can also click right on any variable listed to invoke a menu that provides more documentation on the variable or that allows you to modify the variable.

The extended search characters are available with this command. You can obtain a list of them when you are using this command by typing CTRL-H HELP.

As an example of using the Variable Apropos command (HELP V), suppose you want to find all the variables that contain the string *max* in them. You press HELP V and type *max*. Figure 3-8 shows the output displayed in the typeout window.

Also refer to the Help commands in paragraph 3.10, Help, Documentation, and Undo Commands.

Figure 3-8 Variable Apropos Command (HELP V)

```
ZWEI variables matching "max".  Variable names are mouse-sensitive.

Kill Or Save Buffers Max Lines:      40.
Max Reset Fraction:      0.2s0
Search Ring Max:      3.
Point Pdl Max:      8.
Flash Matching Paren Max Lines:      200.

Done.
```

Describe Variable

Command

Keystroke: HELP O

Provides documentation about a Zmacs user variable. You are prompted in the minibuffer for the name of a variable (completion available).

Also refer to the Help commands in paragraph 3.10, Help, Documentation, and Undo Commands.

Set Variable

Command

Keystroke: META-X Set Variable

Sets a Zmacs user variable, checking the type of the value. You are prompted in the minibuffer for the name of a variable (completion available). This command displays the current value of the variable and its documentation, and prompts you for a new value. Some checking is performed to see if the value is the correct type.

List Local Variables

Command

Keystroke: META-X List Local Variables

Lists the Zmacs user variables that are local for this buffer.

Make Local Variable

Command

Keystroke: META-X Make Local Variable

Makes a Zmacs user variable local to this buffer. You are prompted in the minibuffer for the name of a variable (completion available). This command makes the variable local to the current buffer. Thus, if you set the variable, you do not affect any other buffer.

Kill Local Variable

Command

Keystroke: META-X Kill Local Variable

Makes a Zmacs user variable global in this buffer. You are prompted in the minibuffer for the name of the variable (completion available). This command makes the variable no longer local to the buffer. Thus, the buffer shares the value with most other buffers.

**Word Abbreviation
Commands**

3.5.4 The Word Abbreviation commands allow you to use word abbreviations, which are single words that Zmacs automatically expands after you type the abbreviation and a space (or any character not a letter or number). A *word* is a contiguous group of letters or numerals that does not contain whitespace. Zmacs contains a mode, called *Word Abbreviation mode*, that is specifically designed for expanding word abbreviations. The Word Abbreviation mode command (META-X Word Abbreviation Mode) activates this mode.

The two kinds of word abbreviations are mode and global. *Mode word abbreviations* work in only one major mode (that is, if you change major mode, they no longer work). *Global word abbreviations* work no matter what the major mode is. If you define an abbreviation that is a mode word for the current mode and is also a global word, the mode word definition takes precedence.

Be sure to make your word abbreviations unique. You should not use common characters that might be a word you use in your text, such as *is*.

When a word abbreviation expands, the capitalization rules in the following table apply:

Capitalization of Word Abbreviation	Capitalization of Expansion
Lowercase	Lowercase
Initial capital	Initial capital
All uppercase	All uppercase if one word; initial capital for each word if more than one word

Add Global Word Abbrev Command

Keystroke: CTRL-X +

Reads the mode abbreviation for words before point. If point is in the middle of a word, this command does not consider the characters after point.

With a negative argument, this command asks you which word abbreviation you want to delete. With a positive argument, this command abbreviates the last n words, where n is the numeric argument; the default is the last 1 word. If there is a region, it is used instead.

Add Mode Word Abbrev Command

Keystroke: CTRL-X CTRL-A

Reads the mode abbreviation for words before point. If point is in the middle of a word, this command does not consider the characters after point.

With a negative argument, this command asks you which word abbreviation you want to delete. (If there is no such mode abbreviation but there is a global, this command asks if it should kill the global abbreviation.) With a positive argument, this command abbreviates the last n words, where n is the numeric argument; the default is the last 1 word. If there is a region, it is used instead.

Define Word Abbrevs Command

Keystroke: META-X Define Word Abbrevs

Defines word abbreviations from the buffer. The buffer typically contains three items to a line: an abbreviation followed immediately by a colon, a usage count (any integer), and a string (double quoted), which is the expansion to use for the abbreviation. For example:

```
cd:      0  "Cynthia Deitz"
ab: (Text) 0  "Artie Briggs"
ba:      0  "Bruce Anderson"
```

You can also specify a mode (Text in this example) after the abbreviation and the colon. It must be enclosed in parentheses, and there should be a space before the opening parenthesis and after the closing parenthesis. The usage count tallies the number of times the abbreviation is expanded. Bad values are ignored.

Edit Word Abbrevs Command

Keystroke: META-X Edit Word Abbrevs

Enters recursive edit on the abbreviation definitions. Press the END key to end the recursive edit.

Expand Only Command

Keystroke: META-CTRL-space bar or META-CTRL-X Expand Only

This command differs depending on which keystroke you use. When using the keystroke META-CTRL-space bar with Word Abbreviation mode turned on, this command expands the last word, but inserts nothing after it. If given an argument, this command causes the computer to beep if the word is not expanded.

You can also use this command on buffers for which Word Abbreviation mode is not turned on by using the keystroke META-CTRL-X Expand Only. This command is not needed for a buffer with Word Abbreviation mode turned on unless you define word abbreviations after having already inserted abbreviations as text and you now need to expand them.

Insert Word Abbrevs Command

Keystroke: META-X Insert Word Abbrevs

Inserts all abbreviations and their expansions into the top of the buffer.

Kill All Word Abbrevs Command

Keystroke: META-X Kill All Word Abbrevs

No word abbreviations are defined after you execute this command.

Kill Global Word Abbrev Command

Keystroke: META-X Kill Global Word Abbrev

Causes the global abbreviation typed to be expunged.

Kill Mode Word Abbrev Command

Keystroke: META-X Kill Mode Word Abbrev

Causes the mode abbreviation typed to be expunged. If there is no mode abbreviation but there is a global abbreviation, this command asks about the global abbreviation.

List Some Word Abbrevs Command

Keystroke: META-X List Some Word Abbrevs

Lists abbreviations or expansions containing the specified string.

List Word Abbrevs Command

Keystroke: META-X List Word Abbrevs

Lists all abbreviations and their expansions.

Make Word Abbrev Command

Keystroke: META-X Make Word Abbrev

Prompts for and makes a new word abbreviation. This command asks for the expansion first, then the abbreviation. With a numeric argument, this command makes a global abbreviation. Otherwise, it makes a local abbreviation for this mode.

Read Word Abbrev File Command

Keystroke: META-X Read Word Abbrev File

Loads the word abbreviation file.

Save Word Abbrev File Command

Keystroke: META-X Save Word Abbrev File

Writes out word abbreviations if changed.

Unexpand Last Word Command

Keystroke: CTRL-X U

Undoes the last expansion only, leaving the abbreviation (that is, if you want the abbreviation left as it is).

Word Abbrev Mode Command

Keystroke: META-X Word Abbrev Mode

Activates a mode for expanding word abbreviations. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Also refer to the Mode commands in paragraph 3.14, Mode and Buffer Attribute Commands.

Word Abbrev Prefix Mark

Command

Keystroke: META-X Word Abbrev Prefix Mark

Marks point as the end of a prefix.

Write Word Abbrev File

Command

Keystroke: META-X Write Word Abbrev File

Writes out all word abbreviations in condensed format. You are prompted in the minibuffer for a file in which to write the abbreviations.

Deleting and Moving Text Commands

3.6 The Deleting and Moving Text commands allow you to delete, move, and copy text. The commands that perform these operations are as follows:

- Delete commands — Allow you to make small deletions such as a character or a line.
- Kill commands — Allow you to put text of any size on the kill history. You can either save or kill the text. If you *save* text, you put it on the kill history without removing it from your buffer. If you *kill* text, you put it on the kill history and delete it from your buffer. You can copy the text on the kill history into a buffer at a later time.
- Mark commands — Allow you to mark a block of text as a *region*. Then you can perform operations on the region such as killing and saving. You can also perform operations such as compiling, evaluating, and printing.
- Register commands — Allow you to save text or cursor locations in a register. At a later time, you can retrieve the text or return to a previous cursor location.
- Yank (Copy) commands — Allow you to copy entries from the kill history into your buffer.

Delete Commands

3.6.1 The delete commands allow you to delete the following items:

- Characters
- Return characters and indentation
- Lines
- Parentheses
- Spaces or tabs

The delete commands, in most cases, do not put what they delete on the kill history as do the kill commands. Thus, when you delete an item, it is not retrievable. The exception is that the Delete Forward (CTRL-D) and the Delete Backward (RUBOUT) commands put what they delete on the kill history if you give them a numeric argument.

The commands List Matching Lines and Occur are related commands (see paragraph 3.18.14, Miscellaneous Search Commands). You can use these commands in conjunction with the commands that delete lines.

Delete Backward

Command

Keystroke: RUBOUT

Deletes one or more characters backward from point. With a numeric argument, this command deletes the specified number of characters and puts the deleted characters on the kill history as one entry.

Delete Blank Lines Command

Keystroke: CTRL-X CTRL-O

Deletes any blank lines after the end of the current line.

Delete Forward Command

Keystroke: CTRL-D

Deletes one or more characters forward from point. With a numeric argument, this command deletes the specified number of characters and puts the deleted characters on the kill history as one entry.

Delete Horizontal Space Command

Keystroke: META-\

Deletes any spaces or tabs on either side of point. With a numeric argument, this command inserts the specified number of spaces.

Delete Indentation Command

Keystroke: META-^

Deletes the return character and any indentation at the front of the line. This command leaves a space in place of these characters where appropriate. With a numeric argument, this command moves down a line first (operating on the end of the current line and the start of the next).

In more informal terms, META-^ concatenates this line to the one above it. With a numeric argument, it concatenates this line to the one below it.

Suppose you execute this command with the cursor in the following position:

```
(setf x
  (cond█ (*permanent-real-line-goal-xpos*))
```

The following line shows the results:

```
(setf x █cond (*permanent-real-line-goal-xpos*))
```

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Delete Matching Lines Command

Keystroke: META-X Delete Matching Lines

Deletes all lines containing the specified string from point to the end of the buffer.

Delete Non Matching Lines Command

Keystroke: META-X Delete Non Matching Lines

Deletes all lines not containing the specified string from point to the end of the buffer.

Delete () Command

Keystroke: META-X Delete ()

Deletes both of the n th innermost pair of parentheses enclosing point, where n is a numeric argument. The default is the first innermost pair of parentheses.

Also refer to paragraph 3.12.4, Parentheses Commands.

Tab Hacking Delete Forward Command

Keystroke: META-X Tab Hacking Delete Forward

Deletes a character forward from point, changing tabs into spaces. A numeric argument specifies the number of characters to delete.

Tab Hacking Rubout Command

Keystroke: RUBOUT in Common Lisp or Zetalisp mode

Keystroke: CTRL-RUBOUT when not in Common Lisp or Zetalisp mode

Rubs out the character before point, changing tabs to spaces. Tabs rub out as if they were spaces. A numeric argument specifies the number of characters to rub out.

Kill Commands

3.6.2 The Kill commands allow you to put text on the kill history and delete it from your buffer. You can also save a region on the kill history without deleting it from your buffer. Section 2 (Deleting and Moving Text) explains kill history operations in detail. The Kill commands allow you to perform the following operations:

- Kill a word backward or forward from point.
- Kill a symbolic expression backward or forward from point.
- Kill a list.
- Kill a line.
- Kill a sentence backward or forward from point.
- Kill a region.

- Put a region on the kill history without deleting it.
- Append one kill to another.

The Kill commands differ from the Delete commands. The Delete commands, in most cases, do not put what they delete on the kill history as do the Kill commands. Thus, when you delete an item, it is not retrievable. The exception is that the Delete Forward (CTRL-D) and the Delete Backward (RUBOUT) commands put what they delete on the kill history if you give them a numeric argument.

Append Next Kill Command

Keystroke: META-CTRL-W

Makes the next Kill command append its text to the previous one, forming one entry in the kill history rather than two. This command allows you to collect text from several places into one block.

Backward Kill Sentence Command

Keystroke: CTRL-X RUBOUT

Kills one or more sentences backward from point. A sentence is ended by a period (.), question mark (?), or exclamation point (!) and followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote ("), single apostrophe ('), right parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.

Backward Kill Sexp Command

Keystroke: META-CTRL-RUBOUT

Kills one or more symbolic expressions backward from point. Suppose you execute this command on the following code:

```
(a b c d e)
```

The following line shows the result:

```
(a b d e)
```

Backward Kill Sexp No Up Command

Keystroke: META-X Backward Kill Sexp No Up

Kills one or more symbolic expressions backward from point. This command works like Backward Kill Sexp unless it encounters an opening parenthesis first.

Suppose you execute this command with a numeric argument of 3 on the following code:

```
(cond ((foo
      (framus)
      bar █
```

The following line shows the result:

```
(cond ((█
```

This command is useful in keyboard macros.

Backward Kill Word

Command

Keystroke: META-RUBOUT

Kills one or more words backward from point.

Kill Backward Up List

Command

Keystroke: META-X Kill Backward Up List

Deletes the list that contains the symbolic expression after point, but leaves the symbolic expression.

Suppose you execute this command on the following code:

```
(a b █(d (g h i) e f) c)
```

The following line shows the results:

```
█(d (g h i) e f)
```

Kill Line

Command

Keystroke: CTRL-K

Kills text to the end of the line. If only blanks and a return character are to the right of the cursor, this command kills them. The first CTRL-K at the beginning of a real line clears the text on the line; the second CTRL-K deletes the blank line.

Deleting consecutive lines saves the entire deleted block as one entry on the kill history. With a numeric argument, this command kills the specified number of lines.

Kill Line Backward

Command

Keystroke: CLEAR INPUT

Kills to the start of the current line. This command does not kill the text after point on the current line.

Kill Region

Command

Keystroke: CTRL-W

Kills all text from point to mark (that is, the region). The killed text is placed on the kill history for retrieval.

You can also execute this command by double clicking middle twice with the mouse after marking the region. Refer to paragraph 3.15, Mouse Commands.

Kill Sentence

Command

Keystroke: META-K

Kills one or more sentences forward from point. A sentence is ended by a period (.), question mark (?), or exclamation point (!) and followed by two spaces or a return character (with optional space), with any number of closing characters between sentences. Closing characters are double quote ("), single apostrophe ('), right parenthesis ()), and right square bracket (]). A sentence also ends before a blank line.

Kill Sexp

Command

Keystroke: META-CTRL-K

Kills one or more symbolic expressions forward from point.

Kill Sexp No Up

Command

Keystroke: META-X Kill Sexp No Up

Kills one or more symbolic expressions forward from point. This command works like Kill Sexp unless it encounters a closing parenthesis first. In this case, it stops.

This command is convenient when you do not know how many symbolic expressions are in the list but you want to delete all of them. In this case, you supply a large numeric argument with this command. Then, all the symbolic expressions in the list are deleted, but no symbolic expressions beyond the list are deleted. This command is useful in keyboard macros.

Kill Word

Command

Keystroke: META-D

Kills one or more words forward from point.

Save Region Command

Keystroke: META-W

Puts the region on the kill history without removing it from your buffer.

Mark Commands 3.6.3 The Mark commands allow you to perform the following operations:

- Mark the beginning of the buffer
- Mark a definition such as a **defun**
- Mark the end of the buffer
- Mark a page
- Mark a paragraph
- Mark a symbolic expression
- Mark a quantity
- Set the mark
- Exchange point and mark

You can also mark a region or an item by using the mouse, as shown in the following table:

Command	Mouse Clicks
Mark Region	Hold down the left mouse button and move the mouse until the region you want is marked.
Mark Item	Click middle.

For more information on these commands, refer to paragraph 3.15, Mouse Commands.

Section 2, Zmacs Operations, explains marking in detail.

Abort One Level Command

Keystroke: CTRL-G

Unmarks a region, corrects editing errors, clears the minibuffer for a partially entered command (one not executing yet), and deletes prefixes such as CTRL-X. If nothing is pending, this command only makes the screen flash. With a numeric argument, this command kills a region when inside a keyboard macro without killing the macro and without making the screen flash.

Also refer to paragraph 3.10.1, Abort Commands.

Mark Beginning Command

Keystroke: CTRL-<

Puts mark at the beginning of the buffer.

Mark Definition Command

Keystroke: META-CTRL-H

Puts point and mark around the current definition. The definition can be any defining construct that begins with an opening parenthesis in the first column followed by a `def`, such as `defflavor`, `defmethod`, and `defvar`.

Mark End Command

Keystroke: CTRL->

Puts mark at the end of the buffer.

Mark Page Command

Keystroke: CTRL-X CTRL-P

Puts point at the top of the page and mark at the end. A numeric argument specifies the page: 0 for current, 1 for next, -1 for previous, and larger numbers to move several pages. The default is the current page. A new page is started by any line whose first character is one of the elements of the Zmacs user variable Page Delimiter List.

Mark Paragraph Command

Keystroke: META-H

Sets point and mark around the current paragraph. Paragraphs are delimited by blank lines or by lines that start with a delimiter in the Zmacs user variable Paragraph Delimiter List or in the user variable Page Delimiter List. If there is a fill prefix, any line that does not start with the fill prefix starts a paragraph. If a line starts with a character that is in the user variable Text Justifier Escape List and that is also in the user variable Paragraph Delimiter List, the line counts as a blank line because it separates paragraphs and is not part of them. You can see and change the values of these variables by executing the Variable Apropos command (HELP A).

Mark Sexp Command

Keystroke: META-CTRL-@

Sets mark one or more symbolic expressions from point. With a numeric argument (positive or negative), this command sets mark the specified number of symbolic expressions from point. The default is 1.

Mark Whole Command

Keystroke: CTRL-X H

Puts mark at the beginning of the buffer and point at the end. With a numeric argument, this command does the opposite.

Mark Word Command

Keystroke: META-@

Sets mark one or more words from point. With a numeric argument (positive or negative), this command sets mark the specified number of words from point. The default is 1.

Set Pop Mark Command

Keystroke: CTRL-space bar

Sets or pops mark in the following ways:

- Without a CTRL-U argument, this command sets mark at point and pushes point onto the point PDL.
- With one CTRL-U, this command moves point down the point PDL. (The top entry on the point PDL becomes point.) Each pressing of CTRL-U CTRL-space bar moves point farther down the point PDL, leaving mark at the original position for easy return with the Swap Point and Mark command (CTRL-X CTRL-X).
- With two CTRL-Us, this command pops the point PDL and discards it. The top entry on the point PDL becomes point, and this entry is discarded from the point PDL.

Also refer to paragraph 3.4.4, Saving Cursor Locations (Point) Commands.

Swap Point and Mark Command

Keystroke: CTRL-X CTRL-X

Exchanges point and mark.

Also refer to paragraph 3.4.4, Saving Cursor Locations (Point) Commands.

Register Commands

3.6.4 The Register commands allow you to store text and cursor locations (points) in registers. Then you can yank the text or restore (return to) cursor locations. If you want, you can store a cursor location and text simultaneously in a register. Section 2, Zmacs Operations, explains register operations in detail. The following list summarizes the operations the Register commands can perform:

- Put text into a register
- Yank text contained in a register
- Kill a register
- Save the current cursor position in a register
- Restore a saved cursor position from a register
- List and display the contents of all registers
- Give a mouse-sensitive display of text saved in registers
- Give a mouse-sensitive display of positions saved in registers
- Display the contents of a register

Registers have one-letter names. For example, you can have a register named A, B, C, or even \$, as long as it is a single character. You cannot specify a character with attribute bits (that is, a key used in combination with SUPER, HYPER, META, or CTRL). The limit on the number of registers you can create is the number of characters available to name them.

Put Register

Command

Keystroke: CTRL-X X

Puts text from the region into a register. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits. With an argument, the text is also deleted.

Save Position

Command

Keystroke: CTRL-X S

Saves the current point in a register. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits.

The following paragraphs describe the commands to list and view registers. Some of these commands allow you to use their display to yank the text or restore the cursor location from the register.

List Registers

Command

Keystroke: META-X List Registers

Lists and displays the contents of all defined registers. This command shows both point and text. It is not mouse-sensitive. The Show All Registers command is mouse-sensitive but shows only text.

Figure 3-9 provides an example of the output of the List Registers command. When you see [EMPTY], no text is saved in the register.

Figure 3-9 List Registers Command

```

List of all registers:
C      by the peo ... the earth.
B      [EMPTY]
      Position: Buffer GETTYSBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; LAM11:,
      We have co ... ld as a fi -|- nal restin ... here gave ...
A      But, in a ... poor powe
      Position: Buffer GETTYSBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; LAM11:,
      advanced. ... ense, we c -|- annot dedi ... create--we ...
Done.

```

Show All Registers

Command

Keystroke: META-X Show All Registers

Provides a mouse-sensitive display of text saved in registers with the Put Register command (CTRL-X X). This command shows only text. The List Registers command shows both text and points, but it is not mouse-sensitive.

Figure 3-10 provides an example of the output of the Show All Registers command.

Figure 3-10 Show All Registers Command

```

This is a mousable list of saved text which you may yank.
  Save text with Control-X X plus a register-name letter.

Text in Register C:
<< Yank the entire register >>
by the people, for the people,
shall not perish from the earth

Text in Register A:
<< Yank the entire register >>
But, in a larger sense, we cannot dedicate--we cannot consecrate--we
cannot hallow--this ground. The brave men, living and dead, who
struggled here have consecrated it, far above our poor powe

```

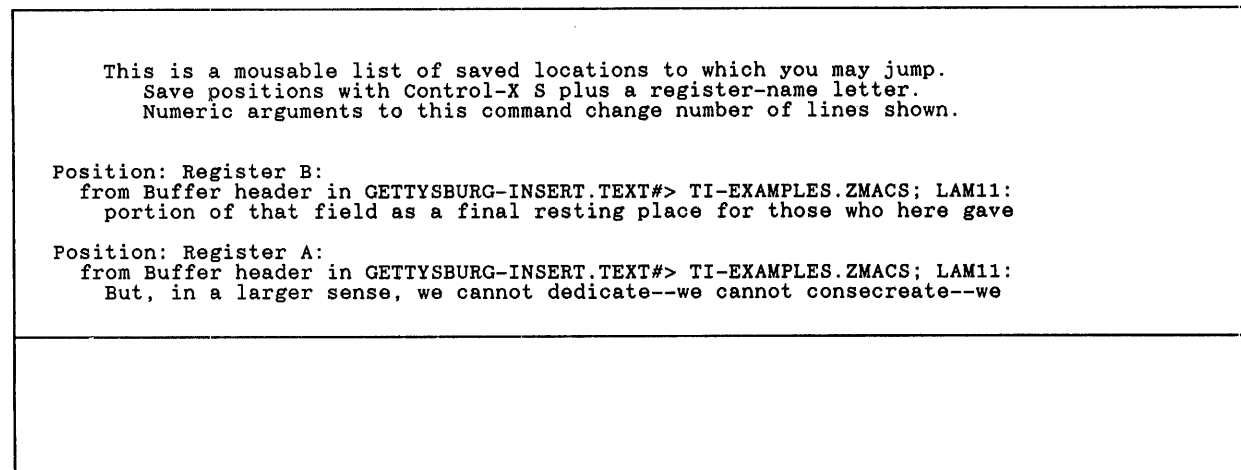
Show Saved Positions

Command

Keystroke: META-X Show Saved Positions

Provides a mouse-sensitive display of positions saved in registers with the Save Position command (CTRL-X S). This command pushes the current point on the point PDL for easy return with various commands. To return to a saved position, you click on the line of text.

Figure 3-11 provides an example of the output of the Show Saved Positions command.

Figure 3-11 Show Saved Positions Command

View Register

Command

Keystroke: META-X View Register

Displays the contents of a register: cursor locations and/or text. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits. The display is mouse-sensitive. If you click on point (if any), you are taken there. You can yank the text (if any) in its entirety or on a line-by-line basis. You can control the insertion of carriage returns after yanking a single line by using the Show Kill History display (CTRL-STATUS) or by changing the value of the Zmacs user variable CR After Line Mouse Yank. This variable is accessible with the Variable Apropos command (HELP-V).

Figure 3-12 provides an example of the output of the View Register command.

Figure 3-12 View Register Command

```

REGISTER A:
  Position: Register A:
    from Buffer header in GETTYSBURG-INSERT.TEXT#> TI-EXAMPLES.ZMACS; LAM11:
      But, in a larger sense, we cannot dedicate--we cannot consecrate--we

Text: Register A:
  << Yank the entire register >>
  But, in a larger sense, we cannot dedicate--we cannot consecrate--we
  cannot hallow--this ground. The brave men, living and dead, who
  struggled here have consecrated it, far above our poor powe

```

The following paragraphs describe the commands that yank text or restore a cursor location from a register.

Get Register

Command

Keystroke: META-X Get Register

Inserts text contained in the specified register. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits. This command leaves point before, and mark after, the text. With an argument, this command puts point after, and mark before, the text.

If a series of blank lines occurs where you want to insert the text, this command moves the blank lines down before inserting the text. This operation differs from the Open Get Register command, which inserts the text into the blank lines, leaving one blank line after the inserted text.

Jump to Saved Position

Command

Keystroke: CTRL-X J

Restores a saved position from a register. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits.

Also refer to paragraph 3.4.4, Saving Cursor Locations (Point) Commands.

Open Get Register

Command

Keystroke: CTRL-X G

Inserts text from a specified register, deleting blank lines. You are prompted in the minibuffer for the register name, a character with no attribute bits. This command leaves point after, and mark before, the text. With an argument, it puts point before, and mark after, the text.

If a series of blank lines occurs where you want to insert the text, this command inserts the text into the blank lines, leaving one blank line after the inserted text. This operation differs from the Get Register command, which moves the blank lines down before inserting the text.

You use the following command to delete a register:

Kill Register

Command

Keystroke: META-X Kill Register

Kills a register. You are prompted in the minibuffer for the register name, which must be a character with no attribute bits. The text of the register is destroyed.

**Yank (Retrieve)
Commands**

3.6.5 The Yank commands allow you to yank (retrieve) an entry from the kill history into your buffer. You can yank all or parts of an entry. A save followed by a yank is the way Zmacs allows you to *copy* text. A kill followed by a yank is the way Zmacs allows you to *move* text. Section 2, Zmacs Operations, explains how to yank text. The following list summarizes the operations you can perform with the Yank commands:

- Show the kill history.
- Yank the last entry that you killed.
- Rotate through the kill history with yanks.
- Use the mouse to kill and yank.

The Copy From Previous Line command allows you to copy text from the previous line. It is not actually a Yank command, but it does perform a copy operation.

Copy From Previous Line

Command

Keystroke: META-X Copy From Previous Line

Copies characters from the last nonblank line. A numeric argument specifies the number of characters.

Pop Minibuffer History Command

Keystroke: META-CTRL-Y in the minibuffer

Cycles backward through the previous minibuffer commands and the data they were given when entered.

Also refer to paragraph 3.10.6, Minibuffer Help Commands.

Show Kill History Command

Keystroke: CTRL-STATUS

Shows the complete contents of the kill history. This display is mouse-sensitive. You can yank entire kills or individual lines of the kills. You can yank as many times as you want.

Yank Command

Keystroke: CTRL-Y

Reinserts the last block of text killed. This command leaves point and mark around what is inserted. Specifying CTRL-U as an argument puts point in front, and mark after, the inserted text. With a numeric argument, this command uses the n th most recent kill from the history, where n is the numeric argument. With an argument of 0, this command lists the contents of the kill history.

You can also execute this command by double clicking middle with the mouse. Refer to paragraph 3.15, Mouse Commands.

Yank Pop Command

Keystroke: META-Y (executed immediately after a CTRL-Y or META-Y)

Corrects a Yank command to retrieve an older entry on the kill history. This command replaces the last entry you yanked with another entry from the kill history. By default, the previous entry is used; thus, successive executions of this command retrieve older and older entries from the history.

A numeric argument specifies which entry on the kill history to retrieve; that is, the argument specifies how many entries forward or backward to go. With a negative argument, this command goes to more recently killed text. With an argument of 0, this command discards the yanked text and replaces it with nothing.

**Directory
Commands**

3.7 The Directory commands allow you to perform a variety of operations dealing with directories:

- Create a directory.
- Edit a directory by using Dired. Dired presents a directory listing and provides a wide range of commands to deal with the files in the directory, such as commands for editing, viewing, deleting, copying, comparing, and printing.
- List the names of all disk directories.
- Clean and expunge a directory.
- Display a directory.

Since Dired contains so many subcommands, it is discussed in a separate paragraph from the other Directory commands.

**Create, List, and
Clean Directory
Commands**

3.7.1 The Create, List, and Clean Directory commands allow you to perform the following operations dealing with directories:

- Create a directory by using the Create Directory command. Dired does not provide this capability, except that if you copy a file to a nonexistent directory, Dired offers to create the directory for you.
- List the names of all disk directories by using the List All Directory Names command. You can also list these names by executing Dired on *host: ~;* where *host* is your local machine.
- Clean a directory and expunge a directory by using the Clean Directory and Expunge Directory commands, respectively. Dired contains similar commands.
- Display a directory. Several commands display directories and provide various options. However, they do not provide the wide range of capabilities that Dired does. These commands are the Display Directory, List Files, View Directory, and View Login Directory commands.

Clean Directory

Command

Keystroke: META-X Clean Directory

Deletes excess versions from superfluous files in the specified directory. See paragraph 3.7.2.13, Dired Hog Commands, for information about the way this command works.

Create Directory

Command

Keystroke: META-X Create Directory

Creates a directory.

Display Directory Command

Keystroke: CTRL-X D

Displays the directory entries for all versions of the current buffer's file. This command uses the directory listing function in the Zmacs user variable Directory Lister. With an argument, this command accepts the name of a file whose directory you want to list.

Expunge Directory Command

Keystroke: META-X Expunge Directory

Expunges deleted files from a directory.

List All Directory Names Command

Keystroke: META-X List All Directory Names

Lists the names of all directories for a specified host. You are prompted in the minibuffer for the name of the host. The directory names are mouse-sensitive. You can click left to see the Dired listing for the directory your mouse cursor is on or click right to access a pop-up menu consisting of the commands View and Edit, which are mouse-sensitive.

List Files Command

Keystroke: META-X List Files

Provides a brief directory listing with several files listed on each line.

The filenames are mouse-sensitive. Clicking left finds the file so that you can edit it. Clicking right invokes a menu of commands that consists of Load, Find, and Compare. *Compare* means to compare the file with the latest version of the file.

View Directory Command

Keystroke: META-X View Directory

Lists the contents of a directory. While viewing the directory, you can use CTRL-V and META-V to scroll forward and backward, respectively. Press END or RUBOUT to exit.

View Login Directory Command

Keystroke: META-X View Login Directory

Lists the files in your directory.

Edit a Directory (Dired) Commands

3.7.2 The directory editor (Dired) presents a listing of a directory and provides a wide range of commands to deal with the files in the directory, such as commands for editing, viewing, comparing, copying, and printing. When you are in Dired, you are in Dired mode. Dired commands are designed to be used only within Dired. Figure 3-13 shows a sample Dired listing.

Figure 3-13

Example Dired Listing

```

Romeo.t1-7: LAURIE; *.*##
Free=1829, Reserved=456, Used=27715 (55 pages used in LAURIE;)
54 blocks in files listed
BABYL.TEXT #1 1 301(8) | 09/18/86 13:47:14
BABYL.TEXT #2 31 31562(8) | 03/19/87 10:11:36
BABYL.TEXT #3 21 21223(8) | 03/19/87 10:12:23
BUG-INIT.TEXT #1 1 142(8) | 03/12/87 15:40:23
GETTYSBURG-ADDRESS.TEXT #1 0 0(8) | 03/22/87 15:39:16
File

```

The information on the listing is as follows:

- The name of the directory.
- The line that begins with Free tells you the status of the blocks that make up a file band:
 - Free — Number of blocks available in the file band.
 - Reserved — Number of blocks for files that are deleted but not expunged. Also, some blocks are reserved by the file system.
 - Used — Number of blocks used in the file band.
 - Unusable — Blocks that cannot be read or written to. These blocks are never allocated. (This status does not appear in the example.)
- The names of the files and subdirectories within the directory. The columns after the names provide the following information:
 - The version number.
 - The length of the file in blocks.
 - The length of the file in bytes, followed by the length of the byte in bits (in parentheses).
 - An exclamation mark (!) if the file has not been backed up to tape. An at-sign (@) if the file is marked against deletion. A dollar sign (\$) if the file is marked against reaping. A number sign (#) if the file is marked against being superseded.
 - The creation date and time.
 - Optionally, the last date the file was read.
 - Optionally, the author.

The directory listing allows you to see what files and subdirectories you have. You can enter a Dired command by typing the character for the command next to the filename. The keyboard cursor must be on the line that contains the name of the file for which you want to execute one of the commands. However, you can enter the commands in any column of the line. The character for the command appears in the first column.

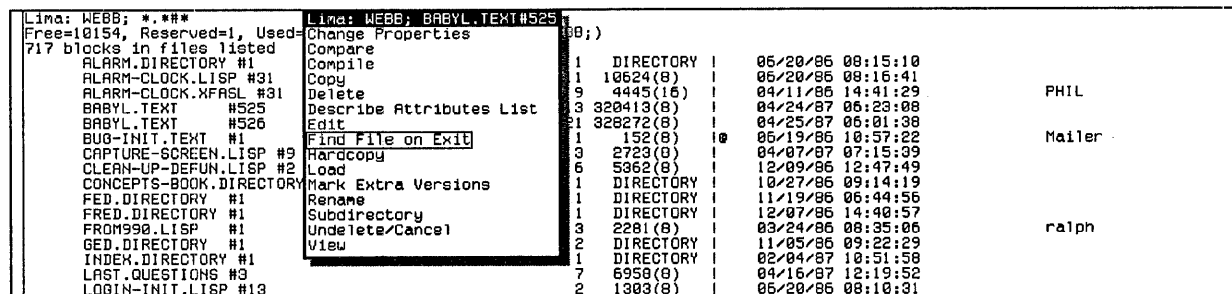
If you want to edit a file, for example, you can type E beside the filename, and Zmacs puts you in a buffer containing a copy of the file. The E command takes effect immediately. Some of the other commands do not. For example, the D command marks a file for deletion with a D. Nothing happens to the file until you are ready to exit Dired with the Q or X command. Then you are asked if you want to delete the file.

When you go to another buffer (for example, by typing E beside a filename), Dired tells you that META-CTRL-L returns you to Dired. META-CTRL-L is the Select Previous Buffer command, which always returns you to the previous buffer whether you are in Dired or not. Dired merely tells you about META-CTRL-L so that you can return to Dired.

If you press HELP in Dired, you receive one more option than in the standard Help menu (which is invoked by pressing HELP in the editor buffer window). This option is the first one listed: M. It provides information on all the special commands available in Dired.

Dired also provides a menu from which you can execute many of its commands. When the mouse is pointing to an entry in the Dired listing, you can click middle to invoke a menu of commands to operate on the entry. Figure 3-14 shows this menu.

Figure 3-14 Dired Menu



The paragraphs that follow discuss the Dired commands in the following groups:

- Dired command — Allows you to enter Dired.
- Dired Documentation command — Provides help on Dired commands.
- Dired Abort command — Aborts Dired.
- Cursor Movement commands — Allow cursor movement in Dired.
- Numeric arguments — Entered with commands in Dired.
- Dired Edit commands — Allow you to edit files or execute Dired on a directory.

- Dired Find File command — Marks a file to be read into the editor.
- Dired View File command — Allows you to view a file
- Dired Delete command — Allows you to delete files.
- Dired Print File command — Allows you to print files.
- Dired Apply Function command — Allow you to perform a function application on a group of files.
- Dired Undelete commands — Allow you to remove the mark from a file marked for an action such as deletion.
- Dired Hog commands — Allow you to find and delete files with excess versions.
- Dired Execute commands — Allow you to perform the actions (such as deleting) on a group of files that you marked.
- Dired Rename command — Allows you to rename a file.
- Dired Copy command — Allows you to copy a file.
- Dired Compile and Load File commands — Allow you to compile a file on disk and load a file so that you can use the functions defined in it.
- Dired Subdirectory command — Allows you to list all the files in a subdirectory.
- Dired Next Undumped command — Allows you to find the next file not backed up to tape.
- Dired Flag commands — Allow you to change the flags associated with files.
- Dired Print File Attributes command — Allows you to print the attributes of a file.
- Dired Change File Properties command — Allows you to change the properties of a file.
- Dired SRCCOM command — Allows you to compare a file with the highest version number of the file.
- Dired Sort commands — Allows you to sort the Dired listing.
- Dired Mouse commands — Allow you to invoke a menu of Dired commands for a file.

NOTE: These Dired commands are valid only after you enter Dired. When you execute a Dired command, you do not need to enter most of the command names with META-X, such as META-X Dired Apply Function. You only need to type the character (such as A) beside the filename in the listing.

Dired Enter Commands 3.7.2.1 The Dired command (CTRL-X CTRL-D) allows you to enter Dired.

Dired Command

Keystroke: CTRL-X CTRL-D

Edits a directory. For documentation on the Dired commands, enter Dired, press HELP, and then press M.

You can also enter Dired from outside the editor by using the **dired** function call. Refer to the Entering Zmacs paragraph in Section 2, Zmacs Operations.

Dired Documentation Command 3.7.2.2 The Dired Documentation command (HELP) provides documentation on the Dired editor and on commands.

Dired Documentation Command

Keystroke: HELP

Displays various kinds of editor documentation. The Help command displays a list of single-letter commands. For example, pressing M after HELP displays documentation on the commands available for Dired.

Dired Abort Command 3.7.2.3 The Dired Abort command allows you to abort Dired. This command returns you to the previous buffer. The Dired buffer remains even after you abort Dired. You can return to the Dired buffer when you execute the List Buffers command (CTRL-X CTRL-B).

If you marked a file for an operation such as D for deletion, this command does not perform the operation, whereas the Q or X command asks you whether you want to perform the operation. If you return to the Dired buffer after pressing the ABORT key, the files are still marked.

Dired Abort Command

Keystroke: ABORT

Aborts Dired.

Cursor Movement Commands 3.7.2.4 Most of the cursor movement commands work the same way in Dired as they do in the editor buffer. You can use the following methods to move the cursor in Dired:

- CTRL-P and ↑ — Take you to the previous line.
- Space bar, CTRL-N, and ↓ — Take you to the next line.

- Mouse — Positioning the mouse cursor beside a filename and clicking left moves the keyboard cursor there.
- CTRL-↑ — Takes you to the previous page.
- CTRL-↓ — Takes you to the next page.

Numeric Arguments

3.7.2.5 Numeric arguments work for almost all Dired commands. For example, if you mark a group of files for deletion, you can give an Undelete command a numeric argument to remove the mark from the number of files you want. (Files are not immediately deleted. They are only marked for deletion.) If you have six files marked for deletion, you can press CTRL-6 U. Six files are then undeleted.

Note that the U command considers six *consecutive* files whether they are marked for deletion or not. If the files you marked for deletion are not in consecutive order, you need to enter a numeric argument large enough to include both marked and unmarked files. The command starts with the filename on the line containing the keyboard cursor and moves downward to the next filenames. With a negative argument, the U command starts with the filename on the line above the keyboard cursor and moves upward to the previous filenames.

Dired Edit Commands

3.7.2.6 The Dired Edit commands (E, CTRL-SHIFT-E, and <) allow you to edit files or execute Dired on a directory. If the keyboard cursor is on a line that is a file, you can edit that file. If the line is a directory, you can start a new Dired on that directory. You can also edit the superior directory of the current buffer's directory.

Dired Edit File Command

Keystroke: E

If the keyboard cursor is on a line that is a file, edits that file. If that line is a directory, you start a new Dired.

Dired Edit File Two Windows Command

Keystroke: CTRL-SHIFT-E

Splits the frame into two editor windows and selects the bottom one. The top window shows the original Dired listing. If the keyboard cursor is on a line that is a file, the bottom window allows you to edit that file. If that line is a directory, the bottom window starts a new Dired.

To return to one-window mode, press CTRL-X 1. (This command is the One Window command.) The window that the command displays is the current window (where the blinking cursor is).

Dired Edit Superior Directory Command

Keystroke: <

Edits the superior directory of the current buffer's directory.

Dired Find File Command 3.7.2.7 If you want to edit more than one file, you type F beside each filename. Next, you enter the X or Q command. They prompt you with visit? (Y or N) for each file that you marked with an F. If you answer Y, Dired puts each file in a separate buffer, and you can edit them. Dired does not give you control until it has each file in a buffer. Also, it does not automatically put you in the buffer for the first file that you specify. You need to execute the List Buffers command (CTRL-X CTRL-B) and select the buffer you want.

If you want to start editing one of the files before Dired finishes putting all the files that you marked in buffers, you need to create another editor with SYSTEM-CTRL-E as soon as Dired brings the first file into a buffer. Then, you list the buffers and select the new arrival. Remember, all editor copies share the same buffer list.

Dired Find File Command

Keystroke: F

Marks a file to be read into the editor. This command differs from the E command in that it allows you to bring as many files as you want into buffers for editing and it does not take effect immediately. The E command immediately puts you in a buffer containing the file you marked with an E.

You can remove the F mark with the U command.

Dired View File Command 3.7.2.8 The Dired View File command (V) allows you to view a file.

Dired View File Command

Keystroke: V

Views the current file. You cannot edit the file, but you can see the beginning of the file without waiting for the whole file to be read in. While viewing, you can press the following keystrokes to scroll:

- CTRL-V, CTRL-↓, or the space bar to scroll forward a screen of text
- META-V or CTRL-↑ to scroll backward a screen of text
- CTRL-N or ↓ to scroll forward one line
- CTRL-P or ↑ to scroll backward one line

To exit, press either END or RUBOUT.

Dired Delete Command 3.7.2.9 The Dired Delete command (D) marks files for deletion. It does not actually delete the file. It is one of the commands that does not take effect immediately. It only puts an uppercase D in the first column to mark the file for deletion.

To actually delete (or expunge) the files, you need to enter an X or Q command and answer E (for expunge) to the DELETE? (Y, N, or E) prompt. If you answer Y to the prompt, the files are marked with a lowercase *d*. They are still not expunged. The lowercase *d* is a file system mark that indicates the files are not accessible. (They are logically deleted yet still retrievable with the Undelete command.) Dired does not expunge files marked with a lowercase *d*. To actually expunge the files and return the allocated disk space to free space, you need to execute the Expunge Directory command (META-X Expunge Directory) to expunge the files. Alternatively, you can undelete the files by using the Dired Undelete commands.

Dired Delete Command

Keystroke: D

Marks file(s) for deletion. You can remove the D mark with the U command.

Dired Print File Command **3.7.2.10** The Dired Print File command (P) allows you to print files from Dired.

Dired Print File Command

Keystroke: P

Marks a file to be printed on the standard hard-copy device. You can remove the P mark with the U command.

Dired Apply Function Command **3.7.2.11** The Dired Apply Function command (A) queues a file for function application. If you want to perform an operation on a group of files, you can mark each file with A and then type X or Q. The X command (or the Q command) lists all the files you marked and asks you in the minibuffer for the name of the function to apply to each of the files. The function requires as an argument the *pathname object* for each file, and the returned value is ignored. For example, you can write a function to remove all the nonprintable characters from each file.

Dired Apply Function Command

Keystroke: A

Marks file(s) to have a function applied to them. You can remove the A mark with the U command.

Dired Undelete Commands **3.7.2.12** The Dired Undelete commands (U and RUBOUT) allow you to remove the mark from files that are marked for actions such as D (delete) or F (find). U removes the mark from the file the cursor is on or else from the file above the cursor.

The lowercase *d* is a file system mark. To undelete a file marked with a lowercase *d*, you need to type U, then Q or X, and answer Y to the UNDELETE? (Y or N) prompt.

Dired Reverse Undelete Command

Keystroke: RUBOUT

Unmarks the previous file(s) for action.

Dired Undelete Command

Keystroke: U

Unmarks the next or previous file(s) for action. With a numeric argument, this command operates the specified number of lines downward (or upward with a negative argument). With no argument, this command operates on the current line's file if it is deleted or marked for action; otherwise, this command operates on the previous line.

If a file is marked with an uppercase *D*, the *D* disappears and a space replaces it when you type U. If a file is marked with a lowercase *d* when you type U, a U appears in place of the *d*; then you need to type Q or X and answer Y to undelete the file.

*Dired Hog
Commands*

3.7.2.13 The Dired Hog commands are entered with the keys N and H. N allows you to find the files that have excess versions (hogs), generally three or more. H allows you to mark these hogs, the extra files, for deletion. These files are called *hogs* because they are hogging space on the disk.

The default for the number of files to be saved depends on the file type. The Zmacs user variable Type Hog Alist specifies the default for each file type. Types not in the list default to 2, except for a few types such as temp and memo that default to 0. This variable also controls whether the consecutiveness of the version numbers is important for a file type.

N finds the next file with more than two versions, where the version numbers are consecutive in descending order. For example, if a file has version numbers 284, 283, and 278, it is not found by N because the version numbers are not consecutive. Version number 278 was probably spared from deletion at some previous time because whoever owns the directory wants that file saved. This file, then, is not a hog file, one with excess versions, because it only has the two most current versions and a special version that was saved.

When N finds a file with three consecutive versions, it puts them at the top of the screen on the first row. You can then use H to mark the excess versions for deletion. For example, if you find a file with three consecutive versions, the excess one (lowest version number) is marked for deletion.

Generally, you use these commands after you have accumulated several extra versions of a file.

Dired Automatic Command

Keystroke: H

Marks superfluous versions of the current file for deletion. Superfluous files are those with more numbered versions than the value of the Zmacs user variable File Versions Kept (not counting noncontiguous versions) and those with a type in the user variable Temp File Type List. The user variable Type Hog Alist associates the file type with hog numbers and controls whether the consecutiveness of the version numbers is important for a file type. Files marked with a dollar sign (\$) are always exempted.

Dired Automatic All Command

Keystroke: H with a numeric argument

Marks all superfluous files for deletion.

Dired Next Hog Command

Keystroke: N

Finds the next file with superfluous versions. This is a file with more numbered versions than the value of the numeric argument, if one is supplied, or the entry in the Zmacs user variable Type Hog Alist or in the user variable File Versions Kept. The user variable Type Hog Alist associates the file type with hog numbers and controls whether the consecutiveness of the version numbers is important for a file type.

*Dired Execute
Commands*

3.7.2.14 The Dired Execute commands (Q and X) list the files you have marked for actions and ask you if you want the actions performed or not. These commands perform basically the same operation except that X leaves you in Dired and Q returns you to the previous buffer. They list the files in a group according to how you marked them. For example, if you typed an F beside three filenames, Dired displays the filenames in a group and then the prompt `visit? (Y or N)`.

If you have files marked for deletion and you type an X or Q, the `DELETE? (Y, N, or E)` prompt appears. The following describes what each response means:

- Y — Marks the files with a lowercase *d*. The lowercase *d* is a file system mark. Dired does not expunge files marked with a lowercase *d*. It only expunges files marked in the current session. You need to execute the Expunge Directory command (META-X Expunge Directory) to permanently delete them. Alternatively, you can use the Dired Undelete command (U) to undelete them and start over.
- N — Leaves the files alone. The marks beside the filenames stay.
- E — Deletes and expunges the files.

Dired Exit Command

Keystroke: Q

Leaves Dired, performing the requested deletions, visiting, or printing. This command displays the files to be deleted and/or printed, then asks you to confirm the operation.

Dired Execute Command

Keystroke: X

Performs the requested deletions, visiting, or printing. This command displays the files to be deleted and/or printed, then asks you to confirm the operation.

*Dired Rename
Command*

3.7.2.15 The Dired Rename command (R) allows you to rename a file. When you type R beside a filename, a prompt appears in the minibuffer asking you for a new filename. You can press CTRL-SHIFT-Y to insert the default filename into the minibuffer. Then you can edit the filename.

Dired Rename Command

Keystroke: R

Renames the file on the line containing the keyboard cursor.

*Dired Copy
Command*

3.7.2.16 The Dired Copy command (C) allows you to copy the file to another file. When you type C beside a filename, a prompt appears in the minibuffer asking you for the filename to which you want to copy. You can press CTRL-SHIFT-Y to insert the default filename into the minibuffer. Then you can edit the filename.

Dired Copy Command

Keystroke: C

Copies to a specified filename the file on the line containing the keyboard cursor.

*Dired Compile
and Load File
Commands*

3.7.2.17 The Dired Compile and Load File commands are entered with the keys B and L. B, which uses the `compile-file` function, compiles a file on disk and puts the compiled version on disk with the same filename and a different type, `xld`. To use the compiled code, you then need to load the `xld` file into memory (a separate step).

L allows you to load a file so that you can use the functions (and so on) defined in it. Loading a file means to bring the contents on *disk* into *memory* so that the functions (and so on) can be called. For example, if you define a function `xyz` in a file and then load it, you can actually use the function `xyz` to do what it is supposed to do. Editing a function and typing it into a file does not make the function automatically exist in memory. You must load it, or either compile or evaluate it in the buffer.

Refer to paragraph 3.3, the Compile and Evaluate Commands, for detailed information on compiling and loading files.

The following paragraphs describe the Dired Compile and Load File commands:

Dired Compile File Command

Keystroke: B

Compiles the file on disk and puts the compiled version on disk with the same filename and a different type, `xld`.

Dired Load File Command

Keystroke: L

Loads the current Lisp or `xld` file.

Loading a Lisp file with this command is equivalent to executing the Find File command followed by the Evaluate Buffer command. Loading an `xld` file with this command is equivalent to executing the Load File command.

*Dired Subdirectory
Command*

3.7.2.18 The Dired Subdirectory command (S) allows you to list all the files in a subdirectory.

Directories form a tree structure. A directory can have files and subdirectories in it. Those subdirectories can have files and subdirectories in them. If you execute the Dired command on a directory that has subdirectories in it, Dired produces only a one-line entry for each subdirectory name. It does not automatically display the items in the subdirectory. If you type an S beside the subdirectory name, Dired gives an indented listing of all the items within that subdirectory.

Dired Subdirectory Command

Keystroke: S

Inserts the filenames of this subdirectory into the Dired listing, or removes them if they are already there. If the filenames are not there, they are inserted underneath the line containing the subdirectory name, and they are indented one additional space. You can then delete them, rename them, and so on. If the filenames are already there, this command offers to remove them from the Dired listing. Removing them from the listing does not delete the files. It only makes Dired stop showing them.

Dired Next Undumped Command **3.7.2.19** The Dired Next Undumped command (!) allows you to find the next file or directory that is not backed up on tape.

Dired Next Undumped Command

Keystroke: !

Finds the next file that is not backed up to tape.

Dired Flag Commands **3.7.2.20** The Dired Flag commands (@, #, and \$) allow you to change the flags associated with files. You can mark files to not be deleted, superseded, or reaped. The following list explains each flag:

- Don't Delete (@) — Prevents the file from being deleted.
- Don't Supersede (#) — Prevents newer versions of the file from being created.
- No Reap (\$) — Prevents the deletion of excess versions of the file.

Normally, these flags are turned off. Changing them turns them on. For example, if you change the Don't Delete flag for a file, the file is protected from deletion.

Dired Complement Don't Delete Command

Keystroke: @

Changes the Don't Delete (@) flag. When turned on, this flag prevents the file from being deleted.

Dired Complement Don't Supersede Command

Keystroke: #

Changes the Don't Supersede (#) flag. When turned on, this flag prevents newer versions of the file.

Dired Complement No Reap Flag Command

Keystroke: \$

Changes the No Reap (\$) flag. When turned on, this flag prevents the deletion of excess versions of the file.

Dired Print File Attributes Command **3.7.2.21** The Dired Print File Attributes command (,) prints the attributes contained in the attribute list in the top line of the file. These attributes are mode, font, package, base, and so on. If the file is an xld file, this command also prints the compilation data.

NOTE: If there is no attribute list in the top line of the file, this command does not print anything.

Dired Print File Attributes

Command

Keystroke: ,

Prints the attributes and compilation data of the file on the line containing the keyboard cursor.

Dired Change File Properties Command

3.7.2.22 The Dired Change File Properties command allows you to change the properties of a file. Figure 3-15 shows the menu that this command invokes. The menu contains some sample values. The current value is displayed or highlighted. Table 3-1 describes the purpose of each property.

Figure 3-15

Dired Change File Properties Menu

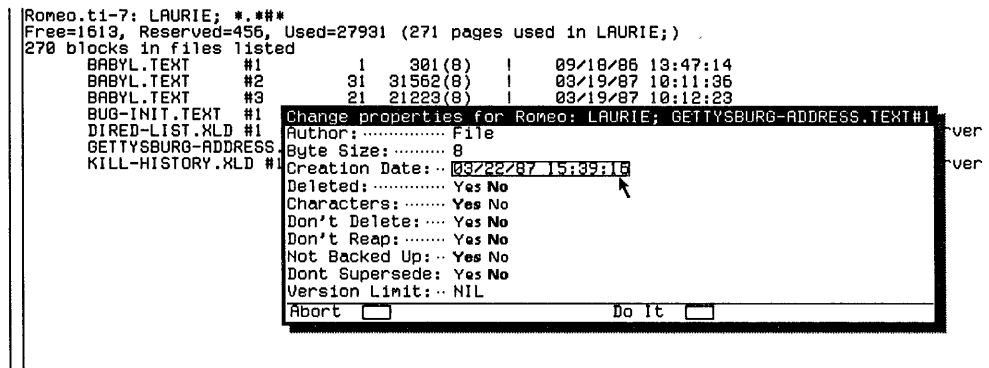


Table 3-1

File Properties

Property	Meaning
Author	Tells who created the file.
Byte Size	The number of bits in one of the basic units used to express the length of the file. (For a text or Lisp file, this basic unit is a character.)
Creation Date	Tells the date and time the file was created.
Deleted	Tells whether the file is marked for deletion.
Characters	Tells whether the file contains characters (such as a text file) or binary data (such as an xld file).
Don't Delete	Tells whether the file is allowed to be deleted.

Table 3-1

File Properties (Continued)	
Property	Meaning
Don't Reap	Tells whether the file is allowed to be reaped.
Not Backed Up	Tells whether the file has been backed up on tape.
Don't Supercede	Tells whether newer versions of the file are allowed.
Version Limit	Tells whether the version number of the file is below the version limit.
<hr/>	
Dired Change File Properties	Command
Keystroke: .	
Changes the properties of the file named on the line containing the keyboard cursor.	
<hr/>	
<i>Dired SRCCOM Command</i>	3.7.2.23 The Dired SRCCOM command (=) compares a file with the newest version of the file. It calls the Source Compare command.
<hr/>	
Dired SRCCOM	Command
Keystroke: =	
Compares the file named on the line containing the keyboard cursor against the version of this file with the highest version number.	
<hr/>	
<i>Dired Sort Commands</i>	3.7.2.24 The Dired Sort commands allow you to sort the Dired listing into one of the following orders:
<ul style="list-style-type: none"> ■ Decreasing or increasing creation date ■ Decreasing or increasing filename ■ Decreasing or increasing reference date ■ Decreasing or increasing size 	
<hr/>	
Sort Decreasing Creation Date	Command
Keystroke: META-X Sort Decreasing Creation Date	
Sorts the Dired listing by creation date (decreasing).	

Sort Decreasing Filename Command

Keystroke: META-X Sort Decreasing Filename

Sorts the Dired listing by filename in reverse alphabetical order.

Sort Decreasing Reference Date Command

Keystroke: META-X Sort Decreasing Reference Date

Sorts the Dired listing by reference date (decreasing).

Sort Decreasing Size Command

Keystroke: META-X Sort Decreasing Size

Sorts the Dired listing by file size (decreasing).

Sort Increasing Creation Date Command

Keystroke: META-X Sort Increasing Creation Date

Sorts the Dired listing by creation date (increasing).

Sort Increasing Filename Command

Keystroke: META-X Sort Increasing Filename

Sorts the Dired listing by filename in alphabetical order.

Sort Increasing Reference Date Command

Keystroke: META-X Sort Increasing Reference Date

Sorts the Dired listing by reference date (increasing).

Sort Increasing Size Command

Keystroke: META-X Sort Increasing Size

Sorts the Dired listing by file size (increasing).

*Dired Mouse
Commands*

3.7.2.25 The Dired Mouse commands enable you to use the mouse to execute some of the commands already described. The following table describes the abbreviations in the mouse documentation window for mouse commands available in Dired.

Abbreviation	Action
L	Moves point.
L2	Moves to point.
M	Displays the file operations menu, a pop-up menu with commands to rename, delete, edit, view, mark extra versions, and so on, which can be used for the file named on the line containing the keyboard cursor.
M2	Displays the Sorting menu, a pop-up menu of commands that enable you to sort by filename, file size, creation date, and so on.
R	Displays the General menu.
R2	Displays the System menu.

File Commands

3.8 The File commands allow you to perform a multitude of operations on a file. The commands are divided into the following groups:

- Change File Properties command — Allows you to change the properties of a file.
- Copy File commands — Allow you to copy one file to another.
- Delete File commands — Allow you to delete a file and set the version limit for a file.
- Find and View File commands — Allow you to find files for editing or viewing.
- Insert File commands — Allow you to append, insert, and prepend to files.
- Print File command — Allows you to print a file.
- Rename File commands — Allow you to rename files.
- Save and Write File commands — Allow you to save a file to disk.

Many of the File commands allow you to use wildcards. A *wildcard* is a substitution for a pathname component that automatically matches all of the names in that component. The character that you use for the wildcard is an asterisk (*).

You can use an asterisk (*) for the whole pathname or any component of the pathname except for the host.

If you want to delete the most recent version of every xld file in a directory, you use the following syntax when you execute the Delete File command (META-X Delete File):

*host: directory; *.XLD*

Refer to the *Explorer Input/Output Reference* for a detailed description of wildcards.

Change File Properties Command

3.8.1 The Change File Properties command enables you to change the properties of a file.

Change File Properties

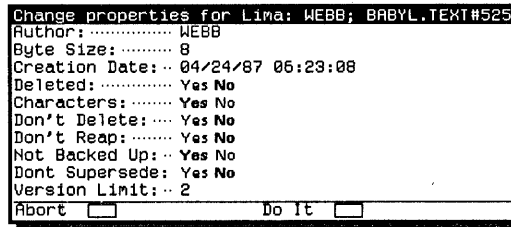
Command

Keystroke: META-X Change File Properties

Allows you to change the properties of a file. Figure 3-16 shows the menu that this command invokes. The menu contains some sample values. The current value is displayed or highlighted.

Figure 3-16

Change File Properties Menu



Refer to Table 3-1 in paragraph 3.7.2.22, Dired Change File Properties Command, for an explanation of the purpose of each property.

Copy File Commands

3.8.2 The Copy File commands allow you to copy files. The following paragraphs describe the Copy File commands:

Copy Binary File

Command

Keystroke: META-X Copy Binary File

Copies one file to another as a binary (not character) file (examples are `xld` and fonts). This command does not copy the creation date or the author unless you specify a numeric argument.

Also refer to paragraph 3.7.2.16, Dired Copy Command.

Copy File

Command

Keystroke: META-X Copy File

Copies one file to another. With a numeric argument of 2 to 6, the command performs the following tasks:

- 2 — Forces copying in character mode.
- 3 — Forces copying in binary mode.
- 4 — Does not copy the creation date or the author.
- 5 — Does not copy the creation date or the author and forces copying in character mode.
- 6 — Does not copy the creation date or the author and forces copying in binary mode.

Copy Text File

Command

Keystroke: META-X Copy Text File

Copies one file to another as a text file. This command does not copy the creation date or the author unless you specify a numeric argument.

Also refer to paragraph 3.7.2.16, Dired Copy Command.

Delete File Commands **3.8.3** The Delete File commands allow you to perform the following operations:

- Delete a file.
- Undelete a file.
- Delete excess versions of a file.
- Set the version limit for a file so that excess versions are automatically deleted.

Delete File Command

Keystroke: META-X Delete File

Deletes a file. If you use wildcards, you can delete many files at once.

Reap File Command

Keystroke: META-X Reap File

Deletes excess versions of the specified file. This command tells you what it intends to delete and save, and it asks for confirmation.

Set Version Limit Command

Keystroke: META-X Set Version Limit

Sets a limit on the number of versions for the specified file or directory. Versions past the limit are expunged (permanently deleted). You are prompted in the minibuffer for the name of a file or directory. The file from which the current buffer came is the default. Then you are prompted for the allowable number of versions. The current limit is displayed.

If you specify the version limit for a directory, the version limit is used as the default for all files in the directory. However, setting the version limit for a single file overrides the directory default.

This command works only on Explorer file systems.

Undelete File Command

Keystroke: META-X Undelete File

Undeletes a file. If you use wildcards, you can undelete many files at once.

**Find and View
File Commands**

3.8.4 The Find and View File commands find files for you. You can either visit or view the file. When you *visit* the file, Zmacs copies the file into a buffer. Once the file is in a buffer, you work on the buffer as if it were the file. If you execute the List Buffers command (CTRL-X CTRL-B) or the Select Buffer command (CTRL-X B) and ask for that file, you are put in that buffer. When you *view* a file, Zmacs allows you only to view the contents of the file. You cannot edit.

Find File

Command

Keystroke: CTRL-X CTRL-F

Visits a file in its own buffer. You are prompted in the minibuffer for a file-name. If the file is already in a buffer, the command selects that buffer. Otherwise, the command creates a buffer whose name is the name of the file, reads the file into that buffer, and selects it.

If you supply a nonexistent pathname, the Zmacs user variable Find File Not Found Is An Error controls whether a new buffer is created or you receive an error message.

Find File No Sectionize

Command

Keystroke: META-X Find File No Sectionize

Visits a file in its own buffer but does not record definitions in the buffer. This command is similar to Find File except that it does not record definitions in the buffer for access with the Edit Definition command (META-.). Furthermore, nothing records such information unless you explicitly execute the Sectionize Buffer command (META-X Sectionize Buffer) on that buffer.

Find System Files

Command

Keystroke: META-X Find System Files

Visits all the files in a specified system. When you type the name of a system defined with `defsystem`, all the files in that system are brought into the editor if they are not already there.

View File

Command

Keystroke: META-X View File

Views the contents of a file. You cannot edit the file, but you can see the beginning of the file without waiting for the whole file to be read in. While viewing, you can press the following keystrokes to scroll:

- CTRL-V, CTRL-↓, or the space bar to scroll forward a screen of text
 - META-V or CTRL-↑ to scroll backward a screen of text
 - CTRL-N or ↓ to scroll forward one line
-

- CTRL-P or ↑ to scroll backward one line

To exit, press either END or RUBOUT.

Visit File Command

Keystroke: CTRL-X CTRL-V

Visits a file in the current buffer, discarding what is already there. You can use this command if you specify the wrong filename when executing the Find File command (CTRL-X CTRL-F) and it fails or you receive a new file. This command is also allowed on a buffer that is not associated with any file (that is, a buffer that does not have the same name as a file stored on disk). In this case, the specified file is read in but not associated with the buffer. This feature is used for reading in old versions of files.

Insert File Commands

3.8.5 The Insert File commands allow you to append, insert, or prepend to files.

Append to File Command

Keystroke: META-X Append to File

Adds the region from your buffer to the end of the specified file. This command creates the file if necessary.

Insert File Command

Keystroke: META-X Insert File

Inserts the contents of the specified file at point. You are prompted in the minibuffer for a filename. This command leaves mark at the end and point at the beginning of the inserted text unless given an argument.

If the inserted file contains different fonts, this command adds the fonts to the buffer's font list. Then, you are asked if you want to change the attribute list (-*-) in the text as well.

Insert File No Fonts Command

Keystroke: META-X Insert File No Fonts

Inserts the contents of the specified file at point but does not insert fonts. You are prompted in the minibuffer for a filename. This command leaves mark at the end and point at the beginning of the inserted text unless given an argument.

If the inserted file contains fonts, the characters that define where each font begins and ends are also inserted. The attribute list is also inserted, if there is one. However, the font characters and the attribute list no longer deal with the fonts in the file; the font characters are treated as text.

Prepend to File Command

Keystroke: META-X Prepend to File

Adds the region to the beginning of the specified file. Creates the file if necessary.

Print File Command 3.8.6 The Print File command allows you to print a file on a hard-copy device.

Print File Command

Keystroke: META-X Print File

Prints a specified file on the local hard-copy device (provided that there is one).

Also refer to paragraph 3.16, Print Commands.

Rename File Commands 3.8.7 The Rename File commands allow you to rename files. You are prompted for a new filename in the minibuffer.

Rename File Command

Keystroke: META-X Rename File

Renames a file. If you use wildcards, you can rename many files at once.

Set Default Filename Command

Keystroke: META-X Set Default Filename

Changes the default filename for most file commands.

Set Visited Filename Command

Keystroke: META-X Set Visited Filename

Changes the file associated with this buffer.

**Save and Write
File Commands**

3.8.8 The Save and Write File commands allow you to write a buffer or region to disk. The Save File and Save All Files commands do not ask you for a file pathname unless there is not one. They default to the file(s) associated with the buffer. The Write File and Write Region commands allow you to specify the file. If you accept the default pathname for Write File, it works like Save File.

Save File

Command

Keystroke: CTRL-X CTRL-S

Writes the current buffer to disk, using the former pathname with an incremented version number. If the current buffer has no file, you are prompted in the minibuffer for a filename. If you made no change in the buffer, this command does not write the buffer to disk.

Save All Files

Command

Keystroke: META-X Save All Files

Offers to write out any changed buffers to their files. You are asked about each file individually. A numeric argument causes the query to be skipped.

NOTE: In addition to this command, the function `zwei:save-all-files` is useful in emergencies when you have modified material in buffers that need to be saved but the editor is malfunctioning. This function does what the Save All Files command does, but it works even when the window system is not working. You execute the function from a Lisp Listener.

Write File

Command

Keystroke: CTRL-X CTRL-W

Writes out the buffer to the specified file. If you accept the default pathname, this command works like Save File.

Write Region

Command

Keystroke: META-X Write Region

Writes out the region to the specified file.

Font Commands

3.9 Each buffer has a set of fonts associated with it. A *font* specifies the typeface and size of characters. The Explorer contains many different fonts. You can use different fonts to highlight different parts of your buffer. You can also create your own font by using the font editor. (Refer to the *Introduction to the Explorer System*.)

Any newly created buffer automatically comes with one default font: `cpfont`. You can then add more fonts and change the default font.

You can put information about the fonts for a buffer in the attribute list at the top of the buffer. The Set Fonts command (META-X Set Fonts) allows you to specify fonts and asks you whether to change the attribute list. When you copy a preexisting file with an attribute list into a buffer, Zmacs uses the fonts that are recorded in the attribute list. Paragraph 2.12, Setting Modes and Buffer Attributes, describes the attribute list in detail. The following line shows an example of an attribute list with fonts:

```
;;--Mode:Common-Lisp;Package:Zwei;Fonts:(medfnt hl12b gothic)--*
```

When you put different parts of your buffer in different fonts, Zmacs inserts internal markers to indicate font changes that are needed when the text is displayed. When the file is written back to disk, these internal markers are changed to epsilon characters followed by a digit. The digit specifies the *n*th font in the attribute list. Zmacs inserts an epsilon followed by an asterisk to return to the previous font. When the file is read back in, these epsilons are changed back into internal font-change markings.

NOTE: Zmacs recognizes an epsilon character as an *unconditional* font-change indicator. If you insert epsilon characters into your file before adding the Fonts: attribute to the attribute list, Zmacs takes the character following the epsilon as a font number whether it is legal or not. Also, these epsilon characters disappear from the file because Zmacs thinks the epsilons are its own markers. You cannot protect epsilons by quoted strings, comments, or even by the `#\` notation. You should not use epsilons and the Fonts: attribute together.

The following list summarizes the operations you can perform with fonts:

- List all the fonts available on the system. You can position the mouse on one of the font names and click left to view the characters in the font.
- Specify up to 26 different fonts for a buffer.
- Change the font of the next character or word.
- Underline a region and change the font of the region.
- Change the default font.
- Use a minor mode to put text in one font, comments in a second font, and quoted strings (commonly, documentation strings) in a third font.
- Compose a complex character not found on the keyboard.

List Fonts Command

Keystroke: META-X List Fonts

Lists the loaded fonts. You can position the mouse on one of the font names and click left to view the characters in the font.

With an argument, this command also lists the files that store the fonts on the system host computer.

Set Fonts Command

Keystroke: META-X Set Fonts

Changes the set of fonts to use. This command sets the fonts associated with the current buffer. You are prompted in the minibuffer for a list of font names, separated by spaces. You can specify up to 26 different fonts for a buffer. This command asks you whether to change the attribute list (-*-) in the text as well.

Also refer to paragraph 3.14, Mode and Buffer Attribute Commands.

Change Default Font Command

Keystroke: META-CTRL-J

Sets the default font for new characters that you type into the buffer. You are prompted in the minibuffer for a letter that signifies the new font from the attribute list (A for the first font and so on).

You can also use one of the following procedures to change the default font after you press META-CTRL-J:

- Click left with the mouse on a character in the buffer to use the character's font.
- Click right to invoke a menu of fonts.
- Press ESCAPE and type a font name. You are then asked whether to change the attribute list (-*-) in the text as well. The new font is added to the end of the set of fonts.

Change Font Char Command

Keystroke: CTRL-J

Changes the font of one or more characters forward (from point). You are prompted in the minibuffer for a letter that signifies the new font from the attribute list (A for the first font and so on).

You can also use one of the following procedures to change the font of the character(s) after you press CTRL-J:

- Click left with the mouse on a character in the buffer to use the character's font.
- Click right to invoke a menu of fonts.
- Press ESCAPE and type a font name. You are then asked whether to change the attribute list (-*-) in the text as well. The new font is added to the end of the set of fonts.

If the previous command was a font-change command, the same font is used for the next font change, without reading any argument.

Change Font Region

Command

Keystroke: CTRL-X CTRL-J

Changes the font of the region. You are prompted in the minibuffer for a letter that signifies the new font from the attribute list (A for the first font and so on).

You can also use one of the following procedures to change the font of the region after you press CTRL-X CTRL-J:

- Click left with the mouse on a character in the buffer to use the character's font.
- Click right to invoke a menu of fonts.
- Press ESCAPE and type a font name. You are then asked whether to change the attribute list (-*-) in the text as well. The new font is added to the end of the set of fonts.

Change Font Word

Command

Keystroke: META-J

Changes the font of one or more words forward (from point). You are prompted in the minibuffer for a letter that signifies the new font from the attribute list (A for the first font and so on).

You can also use one of the following procedures to change the font of the word(s) after you press META-J:

- Click left with the mouse on a character in the buffer to use the character's font.
- Click right to invoke a menu of fonts.
- Press ESCAPE and type a font name. You are then asked whether to change the attribute list (-*-) in the text as well. The new font is added to the end of the set of fonts.

If the previous command was a font-change command, the same font is used for the next font change, without reading any argument.

Change One Font Region

Command

Keystroke: META-SHIFT-J

Changes all characters of one font in a region to another font. You are asked to specify two fonts. Then, this command changes each character in the region that is currently in the first font to the second font instead.

Display Font

Command

Keystroke: META-X Display Font

Displays all the characters in a font.

Electric Font Lock Mode

Command

Keystroke: META-X Electric Font Lock Mode

Activates a minor mode to put text in font A, comments in font B, and documentation strings in font C. (A is the first font listed in the attribute list, B the second, and C the third.) If you supply font D, quoted strings that are not documentation for definitions (fourth in the top-level form) are put in font D; otherwise, they are put in font A. If you supply font E, all function specifications (second in the top-level form) are put in E. Otherwise, they are put in font A. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Also refer to the Mode commands in paragraph 3.14, Mode and Buffer Attribute Commands.

Fontify Region or Buffer

Command

Keystroke: META-X Fontify Region or Buffer

Enables you to use the Electric Font Lock mode on files that are already written. If you do not mark a specific region when you execute this command, the entire buffer is affected.

Compose Character

Command

Keystroke: META-CTRL-C

Composes a complex character from a base character and a diacritic if that character is available in the font. For example, you can create a letter with a tilde over it (\tilde{n}) in most fonts.

You can also type some of these characters from the keyboard. The Symbol Help command (discussed in paragraph 3.10.8) gives the location of these characters.

Help, Documentation, and Undo Commands

3.10 The Zmacs editor provides help or documentation for almost every situation. In addition, it enables you to undo many operations. The following list summarizes the Help, Documentation, and Undo commands:

- Abort commands — Allow you to abort operations in Zmacs.
- Command and Key Help commands — Provide documentation on commands and keys.
- Function and Variable Help commands — Provide documentation on functions and variables.
- List Keystroke History command — Lists the last 60 keystrokes that you typed.
- Minibuffer Help commands — Provide help in the minibuffer, such as pathname completion and listing possible completions for a command name.
- Point Help commands — Provide help on the location of point.
- Symbol Help command — Lists the keystrokes for characters that are not on keycaps, such as \geq .
- Teach Zmacs command — Helps set up the files to be used for the Zmacs editor tutorial.
- Undo commands — Allow you to undo a command you executed.

Section 2, Zmacs Operations, explains the basic operations of the help facilities.

Abort Commands

3.10.1 The Abort commands allow you to perform the following operations in Zmacs:

- Abort a command that is waiting for input.
- Clear the minibuffer for a partially entered command (one not yet executing).
- Unmark a region.
- Abort Dired.
- Correct editing errors.
- Return to the program you were using before you entered the editor (for example, the Lisp Listener).

Abort at Top Level Command

Keystroke: ABORT

Aborts a command that is unfinished or reading arguments. This command aborts minibuffers, recursive edits, and Dired. If you are defining a keyboard macro, this command aborts it.

Abort One Level Command

Keystroke: CTRL-G

Unmarks a region, corrects editing errors, clears the minibuffer for a partially entered command (one not executing yet), and deletes prefixes such as CTRL-X. If nothing is pending, this command only makes the screen flash. With a numeric argument, this command kills a region when inside a keyboard macro without killing the macro and without making the screen flash.

Also refer to paragraph 3.6.3, Mark Commands.

Quit Command

Keystroke: END

Returns to the program you were using before you entered the editor (for example, the Lisp Listener).

**Command and Key
Help Commands**

3.10.2 Zmacs provides documentation on commands and keystrokes. The following provide access to this documentation.

Apropos Command

Keystroke: HELP A

Lists commands whose names contain a given string. Tells which key(s) invoke each command and provides one line of documentation on what the command does. The display is mouse-sensitive. You can execute any command listed by selecting it with the mouse.

The extended search characters are available with this command. You can display a list of these characters when you are using this command by pressing CTRL-H HELP.

Refer to the paragraph 3.1.3 for information concerning minibuffer commands.

- CTRL-/ performs Apropos for the possible completions of what has been typed so far. In other words, CTRL-/ searches for the possible completions of the command name. CTRL-/ looks for any matching string regardless of its position in the command name. Also, the display it provides is mouse-sensitive. You can position the mouse cursor on a command name and click left to execute it.

- CTRL-? provides a menu of the possible completions for the string that has been typed so far. CTRL-? looks only for matching prefix strings (that is, it finds only the commands that begin with the string).

Note that it matters whether the strings you type are plural or singular. If you type `font`, Apropos finds all the commands that contain both `font` and `fonts` since `fonts` is a superstring of `font`. If you type `fonts`, Apropos does not find the commands that contain `font`.

Suppose you want to find all the commands that contain the string `visit`. You press `HELP A` and then type the string `visit`. Figure 3-17 shows the output displayed in the typeout window. Pressing the space bar removes the typeout window.

Figure 3-17 Apropos Command (HELP A)

```

Mini Visited File           Evaluate a form,
  which can be invoked via: Meta-X Mini Visited File
Set Visited Filename       Change the file associated with this buffer,
  which can be invoked via: Meta-X Set Visited Filename
Visit File                 Visit a file in the current buffer,
  which can be invoked via: CTRL-X CTRL-V
Visit Tag Table            Read in the specified tag table file,
  which can be invoked via: Meta-X Visit Tag Table

Done.
```

The Apropos command allows you to make complicated searches by using the CTRL-H prefix operators to locate combinations of strings. For example, you can search for `string1 and string2`, or you can search for `string1 or string2`.

Suppose you want to find all commands that contain the strings `kill` and `Buf`. If you enter `kill Buf` to Apropos, you find only the commands that begin with `Kill Buf`. To find the commands that contain `Kill and Buf`, you need to type `kill`, press CTRL-H, and then type `& Buf`.

Describe Command

Command

Keystroke: `HELP D`

Displays documentation about a Zmacs command that you specify by name. You need type only enough to uniquely identify the command (that is, completion is available).

List Commands

Command

Keystroke: `META-X List Commands`

Lists all extended commands.

Self Document

Command

Keystroke: HELP C

Displays documentation about the command on a given key. You are told what command is associated with a particular key sequence. When you press a key sequence such as META-CTRL-L, this command tells you in the typeout window what META-CTRL-L does. Figure 3-18 shows the output if you press HELP C META-CTRL-L.

Figure 3-18 Self Document Command (HELP C)

```

META-CTRL-L is Select Previous Buffer; implemented by COM-SELECT-PREVIOUS-BUFFER:
Select the previously selected buffer.
A numeric argument selects the argth previous buffer (the default argument
is 2). With an argument of 1, rotates the entire buffer history, and
a negative argument rotates the other way.
This uses the order of buffers that is displayed by List Buffers.

```

The first line indicates that the command is implemented by **com-select-previous-buffer**. If you have source code, you can execute the Edit Definition command (META-.) and type `zwei:com-select-previous-buffer`. (All editor commands are in the ZWEI package.) Then you can read the definition of `zwei:com-select-previous-buffer` and see how META-CTRL-L works.

Where Is

Command

Keystroke: HELP W

Lists all keystrokes that invoke a given command. (This information is given by the Apropos command.) You are prompted in the minibuffer for the command name (completion available).

This command is the reverse of the Self Document command (HELP C), which tells you what command is associated with a particular keystroke.

Suppose you want to find which keystroke invokes the Find File command. You press HELP W and type `Find File`. Figure 3-19 shows what HELP W displays.

Figure 3-19

Where Is Command (HELP W)

```

Find File can be invoked via:  CTRL-X CTRL-F.

```

**Function and
Variable Help
Commands**

3.10.3 The following Zmacs commands provide help on functions and variables.

Arglist Command

Keystroke: META-X Arglist

Lists the arguments for a specified function. You are prompted in the minibuffer for the name of the function. This command displays the result in the minibuffer. If you want to use a function and you do not remember the order of the arguments, this command allows you to find the arguments for the function without reading the source code.

For example, if the function **framus** has been defined in the system by `(defun framus (foo bar) ...)`, the Arglist command tells you that the arguments of **framus** are *foo* and *bar*. If the programmer has provided a return list declaration, Arglist also tells the name of the returned values.

Describe Variable Command

Keystroke: HELP O

Displays documentation for a Zmacs user variable. Zmacs provides *user variables* to allow you to customize many aspects of its workings. You are prompted in the minibuffer for the name of a variable (completion available).

Also refer to paragraph 3.5.3, Variable Commands.

Describe Variable at Point Command

Keystroke: CTRL-SHIFT-V

Displays information about the variable at or before the cursor. The information describes whether the variable is declared special, whether it is bound to a value, and whether it has a documentation string. If none of these are present, this command searches for look-alike symbols in other packages (that is, symbols with the same print name).

Function Apropos Command

Keystroke: META-X Function Apropos

Lists functions containing the given string, using these rules:

- Without a CTRL-U argument, this command searches the current package.
 - With one CTRL-U, this command searches all packages.
 - With two CTRL-Us, this command asks for a package.
-

NOTE: The Function Apropos command is the same as the **apropos** function that you can execute from the Lisp Listener.

The extended search characters are available with this command. You can display a list of them when you are using this command by pressing CTRL-H HELP.

As an example of using the Function Apropos command, suppose you want to search for all the functions in the ZWEI package containing the string `column`. You press CTRL-U CTRL-U META-X, type `Function Apropos`, and press RETURN. Then you type `ZWEI` for the package name, press RETURN, type `column` for the string, and press RETURN. Figure 3-20 shows typical output from this command.

Figure 3-20 Function Apropos Command

```
Functions matching column:
```

```
ZWEI::COM-SET-COMMENT-COLUMN  ZWEI::COM-SET-FILL-COLUMN  ZWEI::COM-SET-GOAL-COLUMN
ZWEI::INDENT-TO-COMMENT-COLUMN  ZWEI::REPORT-COLUMN-SETTINGS
Type CTRL-SHIFT-P to start editing these.
```

You can edit the definitions of the functions this command finds. Pressing CTRL-SHIFT-P takes you to the definition of the first function listed. Each successive CTRL-SHIFT-P takes you to the next function definition. You cannot rotate through the list.

However, this command also creates a **POSSIBILITIES LISTS** buffer, which lists all the functions that it finds. This buffer provides a more permanent listing than the timeout window. You can return to this buffer at any time during your edit session and use it to edit the definitions of the functions.

You can find the **POSSIBILITIES LISTS** buffer by executing the List Buffers command. If you select this buffer, notice that the mode line indicates you are in Possibilities mode and that you can press CTRL-/ to try a possibility (that is, edit the definition of one of the functions). The cursor must be on the line of the function you want to edit.

Figure 3-21 shows the **POSSIBILITIES LISTS** buffer after you execute the Function Apropos command to find all functions in the ZWEI package containing the string `column`. If you execute the Function Apropos command again, it places its results at the beginning of the **POSSIBILITIES LISTS** buffer. The PAGE character separates each execution of the command.

Figure 3-21

Function Apropos Possibilities Buffer

```

Functions matching column:

Edit definition of ZWEI::COM-SET-COMMENT-COLUMN
Edit definition of ZWEI::COM-SET-FILL-COLUMN
Edit definition of ZWEI::COM-SET-GOAL-COLUMN
Edit definition of ZWEI::INDENT-TO-COMMENT-COLUMN
Edit definition of ZWEI::REPORT-COLUMN-SETTINGS

No more functions matching column.

<PAGE>

```

Long Documentation

Command

Keystroke: META-SHIFT-D

Displays long documentation for the specified function. You are prompted in the minibuffer for the name of the function. (The default is the current function from the buffer.) This command displays the arguments and documentation for the function.

Quick Arglist

Command

Keystroke: CTRL-SHIFT-A

Displays the argument list of the function to the left of the cursor. The result is displayed in the minibuffer. If you want to use a function and you do not remember the order of the arguments, this command allows you to find the arguments for the function without reading the source code.

For example, if the function **framus** has been defined in the system by `(defun framus (foo bar) ...)`, the Quick Arglist command tells you that the arguments of **framus** are *foo* and *bar*. If the programmer has provided a return list declaration, Quick Arglist also tells the name of the returned values.

Quick Documentation

Command

Keystroke: CTRL-SHIFT-D

Displays documentation for the function being called from where point is. With a numeric argument, this command prompts you in the minibuffer for the name of the function to document.

Variable Apropos

Command

Keystroke: **HELP V**

Finds all the Zmacs user variables whose names contain a specified string. Zmacs provides *user variables* to allow you to customize many aspects of its workings.

The display given by this command is mouse-sensitive to allow easy modification of variable values. You can click left on any variable listed, and its value appears for editing in the minibuffer. When you have the new value in the minibuffer, press the **END** key to change the variable value.

You can also click right on any variable listed to invoke a menu that provides additional documentation on the variable or that allows you to modify the variable.

The extended search characters are available with this command. You can obtain a list of them when you are using this command by typing **CTRL-H HELP**.

Suppose you want to find all the variables whose names contain the string `max`. You press **HELP V** and type `max`. Figure 3-22 shows the output displayed in the typeout window.

Also refer to paragraph 3.5.3, Variable Commands.

Figure 3-22 Variable Apropos Command (HELP V)

```

ZWEI variables matching "max".  Variable names are mouse-sensitive.

Kill Or Save Buffers Max Lines:      40.
Max Reset Fraction:                  0.2s0
Search Ring Max:                     3.
Point Pdl Max:                       8.
Flash Matching Paren Max Lines:      200.

Done.

```

Where Is Symbol

Command

Keystroke: **META-X Where Is Symbol**

Shows which packages contain the specified symbol.

Also refer to paragraph 3.18.1, Search Commands.

**Help Menu
Commands**

3.10.4 If you press HELP while in the editor buffer window, you invoke a menu of the following commands. These commands are described in their respective groups within paragraph 3.10 of this section. The command that allows you to execute these commands from the Help menu is described below.

Command/Key	Help Command Group
Self Document (HELP C)	Command and Key Help
Describe Command (HELP D)	Command and Key Help
Describe Variable (HELP O)	Function and Variable Help
Apropos (HELP A)	Command and Key Help
Variable Apropos (HELP V)	Function and Variable Help
Undo (HELP U)	Undo
List Keystroke History (HELP L)	List Keystroke History
Symbol Help (HELP S)	Symbol Help
Where Is (HELP W)	Command and Key Help

Documentation

Command

Keystroke: HELP

Executes a specified documentation command. You type one of the following characters:

- To find out what a certain key does, type C and that key.
- To find out what a named command does, type D and the command name.
- To find all commands whose names contain a certain string, type A and then the string.
- To see the last 60 characters you typed, type L.
- Some more advanced options are the following:
 - U — Executes the Undo command.
 - V — Executes the Variable Apropos command.
 - W — Executes the Where Is command.
 - S — Executes the Symbol Help command.
 - O — Executes the Describe Variable command.
 - Space bar — Removes the Help screen.

**List Keystroke
History Command**

3.10.5 The List Keystroke History command provides a list of your last 60 keystrokes.

List Keystroke History Command

Keystroke: HELP L

Lists the last 60 keystrokes that you typed. If you are working in the editor and become unsure about what has happened, this information often helps.

**Minibuffer Help
Commands**

3.10.6 The Minibuffer Help commands provide you with help in the minibuffer.

Evaluate Minibuffer Command

Keystroke: META-ESCAPE

Evaluates a form that you type into the minibuffer.

Also refer to paragraph 3.3.1, Evaluate Minibuffer Command.

Pop Minibuffer History Command

Keystroke: META-CTRL-Y in the minibuffer

Cycles backward through the previous minibuffer commands and the data they were given when entered.

Repeat Minibuffer Command Command

Keystroke: CTRL-X ESCAPE

Repeats a recent minibuffer command. A numeric argument executes the *n*th previous command. The default is 1. An argument of 0 lists the commands that are remembered; these commands are mouse-sensitive.

NOTE: You cannot execute this command if you are already in the minibuffer.

Yank Default String Command

Keystroke: CTRL-SHIFT-Y in the minibuffer

Inserts the default pathname as text into the minibuffer when you are executing one of the File commands.

Where Am I Command 3.10.7 The Where Am I command provides help on the location of point.

Where Am I Command

Keystroke: CTRL-=

Displays information about where point is. This command tells the *x* and *y* positions of point, the octal code for the character following point, and the current line number and its percentage of the total file size. If there is a region, this command tells the number of lines in the region.

Symbol Help Command 3.10.8 The Symbol Help command lists the keystrokes for characters not on keycaps.

Symbol Help Command

Keystroke: HELP S

Lists the keystrokes for characters that are not on keycaps, such as \geq .

Teach Zmacs Command 3.10.9 The Teach Zmacs command sets up the tutorial used in *Explorer Zmacs Editor Tutorial*.

Teach Zmacs Command

Keystroke: META-X Teach Zmacs

Sets up the Zmacs tutorial. This command makes a copy of the Zmacs tutorial files in the user's home directory.

Undo Commands 3.10.10 The Undo commands allow you to perform the following operations:

- Undo your last command.
- Reexecute the command that you undid (that is, undo the undo).

You can use the List Keystroke History Command (see paragraph 3.10.5) to list the last 60 keystrokes that you typed.

Quick Redo Command

Keystroke: CTRL-SHIFT-R

Reexecutes the last command that was undone. You are not queried for approval. Commands that have been undone are remembered for each buffer individually.

Quick Undo

Command

Keystroke: CTRL-SHIFT-U

Undoes the last command that can be undone. You are not queried for approval. Commands that have been undone are remembered for each buffer individually. If there is a region, this command undoes the last batch of changes that occurred within the current region. The region remains so that you can repeat the command on the same region.

Redo

Command

Keystroke: META-X Redo

Reexecutes the last command that was undone in the current buffer.

Undo

Command

Keystroke: HELP U or UNDO

Undoes the last command executed in the current buffer if that command can be undone. If there is a region, this command undoes the last batch of changes that occurred within the current region. The region remains so that you can repeat the command on the same region. This command tells you what it intends to undo and asks for your approval before undoing it.

As an example of using HELP U, suppose you delete two letters from a word in a text file, such as `wh` from `which`. Then you press HELP U. Figure 3-23 shows what HELP U displays.

Figure 3-23

Undo Command (HELP U)

```
Undo small changes (nothing => wh)? (Y or N)
```

The display states that you have `nothing` where the `wh` was because you just deleted it. You answer Y (yes) to put `wh` back where `nothing` is now. The word is restored the way it was.

Within limits, you can also back up to previous changes if you want to undo them. It is also possible to undo the Undo command if you decide to keep your change after all.

Lisp Programming Commands

3.11 The Lisp Programming commands help you develop your Lisp programs. The following list summarizes the operations you can perform with these commands:

- Debug your Lisp program or invoke a Lisp Listener by using the Break command.
- List and edit functions that call other functions by using the Caller commands.
- Edit the definition of a function by using the Edit Source commands. These commands also edit any other defining construct that begins with an opening parenthesis in the first column followed by `def`, such as `defflavor` or `defvar`.
- List and edit flavors and methods by using the Flavor commands.
- Change the package associated with your buffer by using the Set Package command.
- Perform operations with patches by using the Patch commands.
- Edit multiple definitions of a function by using the Possibility commands. If a function is defined in more than one place, each definition is called a *possibility*. These commands also edit any other defining construct that begins with an opening parenthesis in the first column followed by `def`, such as `defflavor` or `defvar`.
- Compare two files or buffers by using the Source Compare commands.
- Trace the execution of a function by using the Trace command.

Break Command

3.11.1 The Break command allows you to debug your Lisp program or invoke a Lisp Listener. When your program is running, you can press the BREAK key in the editor to stop your program. You can create the same effect by putting `(break)` in your program. You can also break before or after a specified function by using the Trace command (META-X Trace). For more information on use of the Break command and the Trace command in Lisp programs, refer to *Explorer Tools and Utilities*.

Break

Command

Keystroke: BREAK

Enters a Lisp break loop.

Caller Commands

3.11.2 The Caller commands allow you to list and edit functions that call other functions. When you change a function, you probably want to see what other functions depend on it because your change might adversely affect the other functions. These commands allow you to find the callers of the function. Then you can test them to see if they still work. If they do not work, you can use these commands to edit the callers.

The Caller commands search through packages, as follows:

- Without a CTRL-U argument, they search the current package.
- With one CTRL-U, they search all packages.
- With two CTRL-Us, they ask you which package you want to search.

For more information on packages, refer to paragraph 3.11.5, Package Commands.

Edit Callers

Command

Keystroke: META-X Edit Callers

Edits the functions that call the specified function.

This command takes you to the definition of the first function that it finds. Pressing CTRL-SHIFT-P takes you through the other definitions. This command also creates a *POSSIBILITIES LISTS* buffer that you can use to edit the functions. In this buffer, you can press CTRL-/ to go to one of the function definitions. The keyboard cursor must be on the line of the function you want to edit. Refer to paragraph 3.11.7, Possibility Commands, for more information on how to use this buffer.

List Callers

Command

Keystroke: META-X List Callers

Lists the functions that reference the specified function or variable.

This command lists the functions that it finds in the typeout window. You can go to the definition of one of the functions by selecting the entry with the mouse (click left). Pressing CTRL-SHIFT-P takes you through the other definitions. This command also creates a *POSSIBILITIES LISTS* buffer that you can use to edit the functions. In this buffer, you can press CTRL-/ to go to one of the function definitions. The keyboard cursor must be on the line of the function you want to edit. Refer to paragraph 3.11.7, Possibility Commands, for more information on how to use this buffer.

Multiple Edit Callers

Command

Keystroke: META-X Multiple Edit Callers

Edits the functions that use the specified functions.

This command is the same as the Edit Callers command except that it continues to ask for callees until you press the RETURN key.

Multiple List Callers

Command

Keystroke: META-X Multiple List Callers

Lists the functions that use the specified functions.

This command is the same as the List Callers command except that it continues to ask for callees until you press the RETURN key.

**Edit or Find
Source Commands**

3.11.3 The Edit Source commands allow you to perform the following operations:

- Edit the definition of a function by using the Edit Definition command.
- Edit the function installed on a key by using the Edit Zmacs command.
- Get a list of the pathname objects for a function by using the Find Source Filename command.

Also, refer to the following related paragraphs:

- List and Edit Changed Definitions Commands, paragraph 3.2.5.
- Caller Commands, paragraph 3.11.2.
- Possibility Commands, paragraph 3.11.7. If the Edit Definition command (META-.) finds more than one definition, these commands help you edit them.

Edit Definition

Command

Keystroke: META-.

Finds the definition of a specified function or special variable and allows you to edit the definition by bringing the file that contains the definition into a buffer. You are prompted in the minibuffer for the name of the function. You can use one of three methods to enter the function name:

- Press RETURN to accept the default, which is the function where the keyboard cursor is.
- Type the name of a function and press RETURN.
- Use the mouse cursor. The mouse highlights with a box any symbol it knows how to find. Click left to find the symbol's definition.

Actually, META-. can find any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defmethod**, **defflavor**, and **defvar**. The defining construct can be user-defined.

If there are multiple definitions, a notification appears in the minibuffer. You can go to successive definitions by supplying a numeric argument to the `META-` command. A list of all definitions is kept in the `*DEFINITIONS*` buffer. In that buffer, the `CTRL-/` command takes you to a definition when the cursor is on the appropriate line. Refer to paragraph 3.11.7, Possibility Commands, for more information.

The Zmacs user variable `Find Patch Definitions Too` allows `META-` to find patch definitions also. The default of this variable is `t`.

Find Source Filename Command

Keystroke: `META-`,

Lists in a typeout window the pathname object for the function at point. This differs from the `Edit Definition` command in that it does not read the file into a buffer.

Edit Zmacs Command Command

Keystroke: `META-CTRL-`.

Edits the function installed on a specified key. This command is similar to the `Edit Definition` command (`META-`), except that instead of supplying the name of a function, you press the key sequence for a command.

Flavor Commands **3.11.4** The Flavor commands allow you to perform the following operations with flavors and methods:

- Describe a flavor.
- Edit or list the component flavors, dependent flavors, direct dependent flavors, or methods of a flavor.
- Edit or list all methods used for a specified operation on a flavor.
- Edit or list all methods of flavors for a specified message.

When two commands (such as `Edit Methods` and `List Methods`) have names that differ only in that one is `Edit-foo` and the other is `List-foo`, the `Edit-foo` command does what the `List-foo` command does and also gives the option of displaying the source in a buffer.

When you execute the `Edit Methods` command and some of the other commands, you can go to the next definition with `CTRL-SHIFT-P` even after intervening operations.

For more information on flavors and methods, refer to the *Explorer Lisp Reference*.

Describe Flavor Command

Keystroke: META-X Describe Flavor

Describes the specified flavor.

Edit Combined Methods Command

Keystroke: META-X Edit Combined Methods

Edits all methods used for a specified operation on a specified flavor.

Edit Flavor Components Command

Keystroke: META-X Edit Flavor Components

Edits the definitions of the component flavors of a flavor.

Edit Flavor Dependents Command

Keystroke: META-X Edit Flavor Dependents

Edits the definitions of the dependent flavors of a flavor.

Edit Flavor Direct Dependents Command

Keystroke: META-X Edit Flavor Direct Dependents

Edits the definitions of the direct dependent flavors of a flavor.

Edit Flavor Methods Command

Keystroke: META-X Edit Flavor Methods

Edits the definitions of the methods of a flavor.

Edit Methods Command

Keystroke: META-X Edit Methods

Edits all methods of flavors for a specified message.

List Combined Methods Command

Keystroke: META-X List Combined Methods

Lists all methods used for a specified operation on a specified flavor.

List Flavor Components Command

Keystroke: META-X List Flavor Components

Lists all the component flavors of a flavor.

List Flavor Dependents Command

Keystroke: META-X List Flavor Dependents

Lists all the dependent flavors of a flavor.

List Flavor Direct Dependents Command

Keystroke: META-X List Flavor Direct Dependents

Lists all the direct dependent flavors of a flavor.

List Flavor Methods Command

Keystroke: META-X List Flavor Methods

Lists all the methods of a flavor.

List Flavor Methods All Command

Keystroke: META-X List Flavor Methods All

Lists all the methods of a flavor, including the component flavors. This command lists the methods that it finds in the typeout window. You can go to the definition of one of the methods by selecting the entry with the mouse (click left).

List Methods Command

Keystroke: META-X List Methods

Lists all methods of flavors for a specified message.

**Set Package
Command**

3.11.5 You can use the Set Package command to change the package associated with your buffer. The new package is used when interning symbols from the buffer. *Interning a symbol* means to either create the symbol or find it if there is a symbol that has the name you typed.

To clarify, the reason that you change package names is to change the variables you are referencing. If a file makes references to variables or functions in the editor (for example), you have the choice of either typing `zwei:` in front of each name or simply putting the file in the ZWEI package and leaving the `zwei:` off all the names. The second method is easier. Another option is to precede each top-level form with `zwei:.`

When you execute the Set Package command, you tell it which package you want. Some of the packages available on the system are the following:

- ZWEI — Zmacs editor package
- USER — Default package when you log in
- W — Window system package
- FS — File system package
- SYS — System internals package

You can also create your own package and specify that package. Refer to the special form `make-package` in the *Explorer Lisp Reference*.

The package that you specify affects only the buffer and not the associated file unless you answer yes to the prompt asking you whether to change the attribute list (-*-) in the text. If you change the attribute list, the package is permanently associated with the file (until you change it). In other words, if you write the buffer to disk, it is associated with the new package when you read in the file again.

Also, you are asked if you want to resectionize the buffer. If you answer yes, all function definitions in the buffer are recorded under symbols in the new package.

One problem may arise if you edit the attribute list. If you type something in the attribute list such as new font information, you must execute the Reparse Attribute List command (META-X Reparse Attribute List) to make the new attribute information take effect. If you do not execute the Reparse Attribute List command, this information is erased when you execute the Set Package command. The Set Package command looks at the properties in the buffer and puts them in the attribute list. The new information you typed is not listed as a property unless you execute the Reparse Attribute List command.

Set Package

Command

Keystroke: META-X Set Package

Changes the package associated with the buffer or file. You are prompted in the minibuffer for a new package to be used when interned symbols that are read from this buffer (for example, when evaluating or compiling parts of the buffer). To specify a package that does not already exist, you must exit with CTRL-RETURN or press RETURN twice. Then you must confirm with yes.

After you specify the package, you are asked whether to change the attribute list (-*-) in the text. If you answer yes, the buffer's attribute list is modified to indicate that the buffer belongs in the new package. This modification affects all operations on the file once you save the buffer. If you answer no to the prompt, the change in package names lasts only for that editing session.

You are then asked whether to resectionize the buffer. If you answer yes, all the function definitions in the buffer are recorded under symbols in the new package.

Also refer to paragraph 3.14, Mode and Buffer Attribute Commands.

Patch Commands

3.11.6 Zmacs provides several commands that perform operations with patches. A *patch* is a small change or set of changes that updates a particular version of a system. Before you make a patch, all existing patches need to be loaded. The *Explorer Lisp Reference* explains patching procedures in detail.

CAUTION: Do not add your own patches to systems released by Texas Instruments.

The Zmacs Patch commands allow you to perform the following operations:

- Start a patch.
- Start a private patch.
- Add the current definition or region to the patch buffer.
- Add the buffer's changed definitions to the patch buffer.
- Add more than one buffer's definitions to the patch buffer.
- Cancel the patch being edited.
- Mark a finished patch as released.
- Resume editing an unfinished patch.
- Finish the current patch file with or without releasing it.

The *patch buffer* is created automatically when you start making a patch. You do not even need to look at it. However, if you want to look at it, you can execute the List Buffers command (CTRL-X CTRL-B) to see it listed.

Start Patch

Command

Keystroke: META-X Start Patch

Begins a patch but does not put any code into it yet. This command asks you for the system to patch. Then it reserves a new version number and creates a patch buffer whose pathname is the source filename for the patch creating the new version number.

This command is similar to the Add Patch command (described later), but it leaves the patch buffer empty except for its initial comments.

Start Private Patch

Command

Keystroke: META-X Start Private Patch

Begins a private patch without associating it with a specific system. This command is similar to the Start Patch command, except that you specify a file-

name for the patch file. The **load-patches** function does not recognize private patches; you must load them explicitly if you want to use them.

Add Patch

Command

Keystroke: META-X Add Patch

Adds the current definition (such as **defun**, **defflower**, or **defconst**) or the region (if any) to the patch buffer. If there is no patch buffer, this command asks you for the system to patch. Then it reserves a new version number and creates a buffer whose pathname is the source filename for the patch creating the new version number. If there is a region, this command appends it to the end of the patch buffer; otherwise, it appends the current definition to the end of the patch buffer.

Add Patch Buffer Changed Definitions

Command

Keystroke: META-X Add Patch Buffer Changed Definitions

Offers to perform the Add Patch command on this buffer's changed definitions. This command does not ask about definitions that have not been changed or that have patched since there were last changed. Type P to patch all the rest of the changed definitions without being asked about each changed definition.

Add Patch Changed Definitions

Command

Keystroke: META-X Add Patch Changed Definitions

Offers to perform the Add Patch command on each changed definition in all buffers. This command does not ask about definitions that have not been changed or that have patched since there were last changed. Type P to patch all the rest of one buffer's changed definitions without being asked about each changed definition. The questions resume in the next buffer.

Cancel Patch

Command

Keystroke: META-X Cancel Patch

Cancels the patch being edited. If this patch is still the most recent one for its system, the patch's version number is reused for the next patch made in the same system.

Release Patch

Command

Keystroke: META-X Release Patch

Marks an already finished patch as released. If you finish a patch with the Finish Unreleased Patch command, the patch is not loaded by **load-patches** unless you specify that you want to test it. After testing it, use this command to mark the patch as released so that users can load it.

Resume Patch

Command

Keystroke: META-X Resume Patch

Resumes editing a patch previously started but never finished. If the patch file source was saved during the previous session, this source is read back in to initialize the editing in this session. You can then proceed to abort the patch or to add to the patch and finish it.

Makes a file containing the names of the files involved in unfinished patches.

Finish Patch

Command

Keystroke: META-X Finish Patch

Finishes the current patch file. This command writes the patch buffer to disk, compiles it, and marks it as finished and released so that **load-patches** loads it. See also the next command, Finish Patch Unreleased.

Finish Patch Unreleased

Command

Keystroke: META-X Finish Patch Unreleased

Finishes the current patch file but does not release it. This command writes the patch buffer to disk, compiles it, and marks it as finished. Since the patch is not released, **load-patches** does not normally load it. To load the patch, answer yes to the special query in **load-patches**; after testing, use the Release Patch command to mark the patch as released so that **load-patches** loads it normally.

Possibility Commands

3.11.7 When you execute a command that finds a definition, such as the Edit Definition command (META-.), it may find more than one definition. A definition can be any defining construct that begins with an opening parenthesis in the first column followed by **def**, **defun** for a function, **defflavor** for a flavor, and so on. When more than one definition is found, the Possibility commands allow you to edit each definition. Each definition is called a *possibility*.

When you execute the Edit Definition command (META-.), it displays the first possibility. It also writes the various possibilities for the definitions into a buffer called ***DEFINITIONS***. If you execute META-. on more than one function, it appends the possibilities for that function to the ***DEFINITIONS*** buffer. You can return to this buffer at any time during your edit session and use it to edit the definitions.

If you execute the List Callers command (META-X List Callers) or the Function Apropos command (META-X Function Apropos), it performs the same kind of operation in a buffer called ***POSSIBILITIES LISTS***. List Callers lists the callers it finds, and Function Apropos lists the functions it finds. You can use the ***POSSIBILITIES LISTS*** buffer to edit the definitions of the callers or functions.

Figure 3-24 shows a sample **DEFINITIONS** buffer after executing the Edit Definition command (META-). To view the **DEFINITIONS** buffer (or the **POSSIBILITIES LISTS** buffer), you can execute the List Buffers command (CTRL-X CTRL-B) and select the buffer.

Figure 3-24 Sample Definitions Buffer

```

Edit definition of zwei:com-find-file
Definitions in buffer COMA.LISP#> ZWEI; L11:
Read in source file COMB.LISP#> ZWEI; L8:
Try other packages, grind definition, or ask for filename

```

When you look at the **DEFINITIONS** buffer, the cursor is at the end of the first entry, and the first definition is in a buffer. The position of point in the **DEFINITIONS** buffer is used to keep track of which possibilities you have already examined. However, you can move point to other lines in the buffer without examining each definition by using the Cursor Movement commands or by positioning the mouse cursor on the line where you want point and clicking left.

You can press CTRL-/ to go to the possibility on the line that contains the cursor. Notice also that the mode line indicates that you can use CTRL-/ to try one of these possibilities. Thus, pressing CTRL-/ takes you to the buffer where the definition is located.

Each possibility is marked as being in a buffer or as being in a file that still needs to be read in. If you press CTRL-/ when the cursor is on the line containing the message *Read in source file*, the file is read into a buffer and that definition is displayed for editing. If you look at the **DEFINITIONS** buffer again, the message *Read in source file* is deleted, and the message *Definitions in buffer* is put in its place.

Go to Possibility Command

Keystroke: CTRL-/

Goes to the possibility on the line in the **DEFINITIONS** buffer or **POSSIBILITIES LISTS** buffer that contains the definition you want to edit.

Go to Next Possibility Command

Keystroke: CTRL-SHIFT-P

Goes to the next possibility (the one after the last one you went to).

Go to Next Top Level Possibility

Command

Keystroke: META-X Go to Next Top Level Possibility

Goes to the next top-level possibility (the one after the last one you went to). In other words, this command goes to the first of the possibilities for the next entry in the file. For example, suppose you execute META-. on two function names. You go into the **DEFINITIONS** buffer and move point into one of several possibilities for the first function. You check that definition, and it is the one you wanted. Then you use this command to go to the possibilities for the next function, skipping the other possibilities for the first function.

Source Compare Commands

3.11.8 The Source Compare commands allow you to perform the following operations:

- Compare two files or buffers.
- Compare a buffer in memory against the buffer's file on disk.
- Compare two files or buffers and merge the differences into a buffer.

Source Compare

Command

Keystroke: META-X Source Compare

Compares two files or buffers. The output is written to the screen and also into a buffer named **SOURCE COMPARE filename or buffer name/ filename or buffer name**.

NOTE: The Dired SRCCOM command (=) invokes the Source Compare command.

Source Compare Changes

Command

Keystroke: META-X Source Compare Changes

Compares a buffer in memory to the buffer's file on disk. If there are no changes, this command clears the buffer's modified bit. The output is written to the screen and also into a buffer named **SOURCE COMPARE of filename or buffer name**.

Source Compare Merge

Command

Keystroke: META-X Source Compare Merge

Allows you to compare two files or buffers and merge the differences into a buffer. This command provides a convenient way to take two versions of a file and to combine and edit all the desired changes.

As an example of why you might use this command, suppose you have one version of a file, and two different people make changes to it. You want to put all the good changes together in one file and leave all the bad points out. This command allows you to do that.

You can compare either files or buffers. The first prompt is as follows:

```
Merge file or buffer: (F or B)
```

You specify whether the first source you are comparing is a file or a buffer. Then a prompt asks you for the pathname. Pressing RETURN specifies the current buffer. Pressing CTRL-SHIFT-F allows you to specify a filename.

Next you are asked if the second source is a file or a buffer. Then you specify a pathname. The next prompt is as follows:

```
Put merged version in buffer
```

You specify the buffer where you want the merged version.

You can compare by form or by text. The following prompt appears:

```
Compare by Text or Forms (T or F)
```

`Text` means that the command compares the lines and determines if they are the same. `Form` means that the command interprets the material as Lisp lists. Each Lisp expression is compared in one file to the Lisp expression in the other file. If they are the same in a Lisp sense, then there is no difference. It does not compare indentation; it just compares meaning. Form comparisons may be substantially slower.

When you compare files or buffers, the command tells you the merged version is the source compare merge of this file (or buffer) versus that file (or buffer). It contains everything that was in either file.

When the information is common to both files, it simply appears in the merge buffer. Sometimes the merge buffer contains the message `MERGE LOSSAGE`. *Lossage* is used in the editor to mean something is wrong, not necessarily that something was lost. Between `***MERGE LOSSAGE***` and `***END OF MERGE LOSSAGE***`, the merge buffer shows the name of each file (or buffer) and the part of the file (or buffer) affected.

Figure 3-25 shows an example merge buffer.

Figure 3-25 Source Compare Merge Command

```

***
*** THIS IS A GOOD EXAMPLE FOR :BINDINGS
*** WILL HAVE TO UPDATE THE TEXT THAT CURRENTLY DESCRIBES HACK HOST.
*** MERGE LOSSAGE ***
*** Buffer BINDINGS.EXAMPLE#> WEBB.WINDOW; Lima: HAS:
(DEFUN BEEP-PLAY ()s
*** File Lima: WEBB.WINDOW; BINDINGS.EXAMPLE#3 HAS:
(DEFUN BEEP-PLAY ()
*** END OF MERGE LOSSAGE ***      ↗
  (BEEP BEEP-TYPE))

(DEFVAR BEEP-TYPE NIL)

(ui:menu-choose '((" " :NO-SELECT)
*** MERGE LOSSAGE ***
*** Buffer BINDINGS.EXAMPLE#> WEBB.WINDOW; Lima: HAS:
  (" SELECT BEEP AND PLAY IT " :FUNCALL BEEP-PLAY
*** File Lima: WEBB.WINDOW; BINDINGS.EXAMPLE#3 HAS:
  ("SELECT BEEP AND PLAY IT" :FUNCALL BEEP-PLAY
*** END OF MERGE LOSSAGE ***
  :BINDINGS ((BEEP-TYPE ',(W:MENU-CHOOSE W:*BEEPING-FUNCTIONS*
                :COLUMN# 2 )))
  :DOCUMENTATION "SELECT A BEEP TYPE FROM A MENU AND PLAY IT")
  (" " :NO-SELECT))
  :LABEL "BEEP MENU"
)

```

Source Compare Merge BINDINGS.EXAMPLE#> WEBB.WINDOW; Lima: vs Lima: WEBB.WINDOW; BINDINGS.EXAMPLE#>
1, 2, *, I, SPACE, RUBOUT, I, Control-R or HELP: █

In the merge buffer, you have the following options for dealing with the differences between the files or buffers:

Keystroke	Description
1	Shows the result of taking the contents of the first source for this difference. To make the change permanent, press the space bar. To cancel the command, press RUBOUT. Press HELP to see other options after the command is entered.
2	Works like 1 but takes the second source for this difference.
*	Leaves both versions for this difference. It only removes the *** lines. The space bar, RUBOUT, and HELP work the same as for 1.
I	Leaves everything, including the *** lines. The space bar, RUBOUT, and HELP work the same as for 1.
space bar	Works like * but no confirmation is needed.

Keystroke	Description
RUBOUT	Deletes both versions for this difference. The space bar, RUBOUT, and HELP work the same as for 1.
!	Takes a second command (1, 2, *, or I) and applies it to the rest of the buffer. For example, if you want to take the 1 option for the rest of the differences in the buffer, you can type !1.
CTRL-R	Lets you edit the buffer at this point. You might want to make a change to one version. To return to merging sources, press the END key. Your changes are now in the buffer.
HELP	Displays a description of these keystrokes.

If you press HELP after entering 1, 2, *, I, or RUBOUT, the command shows other options available at that point, as follows:

Keystroke	Description
Space bar	Confirms the command.
RUBOUT	Cancels the command.
CTRL-R	Allows you to edit the buffer at this point. To return to merging sources, press the END key.
HELP	Provides a description of these keystrokes.

Each time you approve one choice by entering one of these characters and pressing the space bar, the command goes to the next difference in the file. It works down to the end and finishes. Then it *sectionizes* the buffer if there are Lisp definitions in it. The only buffers that have sections are those containing Lisp code. Other buffers do not have sections. A *section* can contain any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defvar**, **defflavor**, and **defun**. The defining construct can be user-defined. Each definition is a section.

After sectionizing the buffer, the command looks at the attribute list (in the top line of the buffer) to see what fonts you have. It then adds these fonts. If the files have different fonts, the command uses the fonts from the set you choose during the merge of the attribute lists.

Trace Command 3.11.9 The Trace command allows you to trace functions.

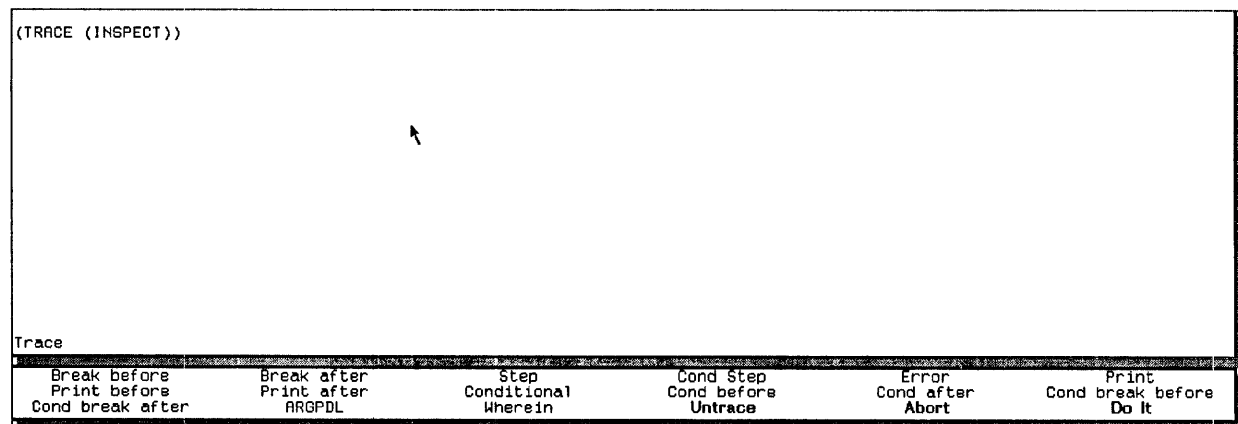
Trace Command

Keystroke: META-X Trace

Allows you to trace functions. When you *trace* a function, certain special actions are taken when the function is called and when it returns. The actions may be simple typeout or more sophisticated debugging functions. The default tracing action is to print one message when the function is called, showing its name and arguments, and another message when the function returns, showing its name and value(s).

The Trace command prompts you in the minibuffer for the name of a function. After you enter the function name, the following menu appears. You can omit the menu step by supplying a numeric argument with this command.

Figure 3-26 Trace Command Menu



To stop tracing the function, you click on Untrace. For more information on this facility, refer to *Explorer Tools and Utilities*.

**Lisp Syntax
Commands**

3.12 The Lisp Syntax commands allow you to manipulate the syntax of Lisp code. The following list summarizes the operations you can perform with these commands:

- Manipulate comments. You can move the cursor by comment line, add semicolons (;) in front of every line in a region, and indent comments.
- Grind (pretty print) a function definition or the evaluation of a form.
- Display the macro expansion of the next symbolic expression.
- Manipulate parentheses. You can move the cursor by parentheses, display the start of a list, find unbalanced parentheses, and move parentheses.

**Comment
Commands**

3.12.1 The Comment commands allow you to perform a variety of operations with comments:

- Add semicolons (;) in front of every line in a region.
- Remove semicolons from lines in the region.
- Fill comments.
- Indent comments.
- Move the cursor up and down comment lines.
- Set the comment column.
- Start and end comments.

Comment Out Region

Command

Keystroke: CTRL-X CTRL-;

Adds a semicolon (;) in front of every line of the region. This command adds semicolons regardless of whether any semicolons are already there. With a positive numeric argument, this command adds the specified number of semicolons to each line of the region. With a negative argument, this command removes semicolons. CTRL-U is treated as -1.

When you mark the region, the underlining must begin on the first character of the first line of the region in order for the command to affect that line. Otherwise, the command ignores the first line of the region.

Uncomment Out Region

Command

Keystroke: CTRL-X META-;

Removes semicolons (;) from lines in the region. This command performs the same operation that the Comment Out Region command performs with a negative argument.

Down Comment Line

Command

Keystroke: META-N

Moves to the comment position in the next line. This command is equivalent to the Down Real Line command (CTRL-N) followed by the Indent for Comment command (CTRL-;) except that any blank comment on the current line is deleted before the keyboard cursor is moved.

Also refer to paragraph 3.4, Cursor Movement Commands.

Fill Long Comment

Command

Keystroke: META-X Fill Long Comment

Fills a multiline comment's paragraphs. *Fill* means to put as much text as possible on the current line without exceeding the right margin. (Refer to paragraph 3.19.2, Fill Commands, for information on setting the right margin.) An entire run of comment lines is filled, each paragraph separately. The comments must begin at the start of the line.

Indent Comment Relative

Command

Keystroke: META-X Indent Comment Relative

Aligns a new comment with the previous one. This command sets the Zmacs user variable Comment Column to the position of the previous comment and then executes the Indent for Comment command (CTRL-;).

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Indent for Comment

Command

Keystroke: CTRL-; or META-;

Moves to or creates a comment. This command finds the start of existing comments or creates a comment at the end of the current line. With a numeric argument, this command realigns existing comments for n lines but does not create any (n is the numeric argument). The following Zmacs user variables affect how this command operates. All units are raster pixels, not character counts.

- **Comment Column** — The minimum column for aligning comments. When you press the CTRL-; keystroke, the keyboard cursor goes to that column. If nothing is there, it goes to the next tab stop. You can set this column by using the Set Comment Column command (CTRL-X ;).
- **Comment Start** — The string used to recognize existing comments.
- **Comment Begin** — The string used to start new comments.
- **Comment Round Function** — The function used to compute the column for comments past the comment column.

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Indent New Comment Line Command

Keystroke: META-LINE FEED

If the keyboard cursor is inside a comment when you press META-LINE FEED, this command continues the comment on the next line and pushes the existing next line down a line to make space for the comment. If the keyboard cursor is not inside a comment, this command acts like the Indent New Line command (LINE FEED), which inserts a return and a tab to obtain the proper indentation on the new line.

Note that this command behaves differently depending on the number of semicolons:

- One semicolon — Indents to the comment column, which you can set with the Set Comment Column command.
- Two semicolons — Indents appropriately for Lisp code.
- Three semicolons — Does not indent.

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Kill Comment Command

Keystroke: META-CTRL-;

Deletes any comment on the current line.

Set Comment Column Command

Keystroke: CTRL-X ;

Sets the Zmacs user variable Comment Column to the current horizontal position. With an argument, this command sets the user variable Comment Column to the position of the previous comment and then aligns or creates a comment on the current line.

Uncomment Region Command

Keystroke: META-X Uncomment Region

Deletes any comments within the region.

Up Comment Line Command

Keystroke: META-P

Moves to the comment position in the previous line. This command is equivalent to the Up Real Line command (CTRL-P) followed by the Indent for Comment command (CTRL-;) except that any blank comment on the current line is deleted before the keyboard cursor is moved.

Also refer to paragraph 3.4, Cursor Movement Commands.

Grind (Pretty Print) Commands 3.12.2 The Grind commands allow you to grind (or pretty print) Lisp code. *Grind* means to indent Lisp code correctly (or to pretty print it). Grind works on a whole Lisp expression, whereas *indent* usually works on one or more lines that you specify or take as the default. You can grind the following:

- Definition of a function
- Evaluation of a form

Grind Definition Command

Keystroke: META-X Grind Definition

Grinds the definition of a function into the buffer. You are prompted in the minibuffer for the name of the function. This command inserts the properly indented definition at point.

Also refer to paragraph 3.16, Print Commands.

Grind Expression Command

Keystroke: META-X Grind Expression

Grinds the evaluation of a form into the buffer. You type a form into the minibuffer. This command evaluates it and inserts the properly indented result at point.

Also refer to paragraph 3.16, Print Commands.

Macro Expansion Commands 3.12.3 The Macro Expansion commands display the macro expansion of the next symbolic expression. A *macro* is a form that translates to another form. For information on macros, refer to the *Explorer Lisp Reference*.

Macro Expand Expression Command

Keystroke: CTRL-SHIFT-M

Displays the macro expansion of the next symbolic expression. The result is displayed on the screen (in the typeout window, not the buffer) using the pretty print format of **grind-top-level**.

Macro Expand Expression All Command

Keystroke: META-SHIFT-M

Displays the macro expansion of the next symbolic expression to all levels. That is, if a macro expansion contains a macro, this command expands the embedded macro also. The result is displayed on the screen (in the typeout window, not the buffer) using the pretty print format of **grind-top-level**.

This command is the equivalent of the Lisp **mexp** function.

**Parentheses
Commands**

3.12.4 Zmacs provides many features that deal with parentheses, such as automatically displaying matching parentheses. When you finish typing the closing parenthesis for an expression, Zmacs flashes the corresponding opening parenthesis. If the cursor is on the opening parenthesis, Zmacs flashes the corresponding closing parenthesis if it exists. Zmacs also provides numerous commands that deal with parentheses.

Some of the options these commands provide are the following:

- Display the start of a list.
- Find unbalanced parentheses.
- Insert closing parentheses.
- Insert matching parentheses.
- Insert matching parentheses backward.
- Move the closing parenthesis forward or backward.
- Move the opening parenthesis backward or forward.

The following commands move the cursor according to parentheses. Refer to paragraph 3.4, Cursor Movement Commands, for a description of these commands.

- Backward Down List (META-X Backward Down List)
- Move Over) (META-))

Close Definition

Command

Keystroke: META-X Close Definition

Inserts enough closing parentheses to terminate this definition (such as a **defun**).

Delete ()

Command

Keystroke: META-X Delete ()

Deletes both of the n th innermost pair of parentheses enclosing point, where n is a numeric argument. The default is the first innermost pair of parentheses.

Also refer to paragraph 3.6.1, Delete Commands.

Find Unbalanced Parentheses

Command

Keystroke: CTRL-(

Finds parenthesis errors in the buffer. Setting the Zmacs user variable `Check Unbalanced Parentheses When Saving` to `t` does this for you automatically every time you save a file.

This command returns one of three possible messages:

- All parentheses appear balanced — Usually indicates there are no errors.
- Probably an extra/missing left/right parenthesis here — Indicates a left or right balancing parenthesis is missing. Point is on the parenthesis in question.
- Cannot find unbalanced parenthesis — Indicates there is an unbalanced parenthesis that cannot be found.

Grow List Backward

Command

Keystroke: META-X Grow List Backward

Moves the opening delimiter of the current list backward over one or more symbolic expressions. With a negative argument, this command shrinks the list by moving the opening delimiter forward. This command marks the beginning of the resulting list for visibility. It always leaves point in a position where the same command with a negative argument undoes the command.

Grow List Forward

Command

Keystroke: META-X Grow List Forward

Moves the closing delimiter of the current list forward over one or more symbolic expressions. With a negative argument, this command shrinks the list by moving the closing delimiter backward. This command marks the end of the resulting list for visibility. It always leaves point in a position where the same command with a negative argument undoes the command.

Make ()

Command

Keystroke: META-(

Inserts matching delimiters, putting point between them, as follows:

- With a positive argument, this command puts the specified number of symbolic expressions within the new set of delimiters by inserting the closing delimiter after the specified number of symbolic expressions.
- With a negative argument, this command puts the specified number of symbolic expressions within the new set of delimiters by inserting the opening delimiter before the specified number of symbolic expressions.

Suppose you execute the command on the following code:

```
(a b c d e f)
```

The following lines show the results:

```
(a b c d (e f))
```

 Without an argument.

```
(a b c d (e f ))
```

 With an argument of 2.

Make () Backward

Command

Keystroke: META-X Make () Backward

Inserts matching delimiters backward. This command performs the same operation that the Make () command performs except that the meaning of the positive and negative arguments is reversed, as follows:

- With a positive argument, this command puts the specified number of symbolic expressions within the new set of delimiters by inserting the opening delimiter before the specified number of symbolic expressions.
- With a negative argument, this command puts the specified number of symbolic expressions within the new set of delimiters by inserting the closing delimiter after the specified number of symbolic expressions.

Suppose you execute the command on the following code:

```
(a b c d e f)
```

The following lines show the results:

```
(a b c (de f))
```

 Without an argument.

```
(a b (c de f))
```

 With an argument of 2.

Show List Start

Command

Keystroke: META-X Show List Start

Displays in the minibuffer the start of the list that point is positioned after.

**Miscellaneous
Commands**

3.13 The Miscellaneous commands are those commands that do not fall neatly into one of the command groups defined by this manual. The operations these commands perform are as follows:

- Count lines in a region or buffer, lines on a page, occurrences of a string, or words in a region or buffer. A *string* is a sequence of characters, not necessarily alphabetic.
- Print the current date into the buffer.

Count Lines

Command

Keystroke: META-X Count Lines

Counts the number of lines in the region or buffer. If no region is marked, this command counts the number of lines in the buffer.

Count Lines Page

Command

Keystroke: CTRL-X L

Counts the number of lines on this page. This command also adds, in parentheses, the number of lines on the page before point and the number of lines after point. (Pages end with a page character, which is created by pressing CTRL-Q CLEAR SCREEN.)

Count Lines Region

Command

Keystroke: META-=

Counts the number of lines in the region. The result appears in the minibuffer.

Count Occurrences

Command

Keystroke: META-X Count Occurrences

Counts occurrences of a specified string after point.

Count Words

Command

Keystroke: META-X Count Words

Counts the number of words in the region or buffer. If no region is marked, this command counts the number of words in the buffer.

Insert Date Command

Keystroke: META-X Insert Date

Inserts the current date into the buffer. This command calls **time:print-current-time**, or if given an argument, it calls **time:print-current-date**.

Various Quantities Command

Keystroke: CTRL-Q

Given characters with attribute bits or nonletters, this command inserts them. A question mark (?) displays documentation about this command.

Given octal digits, this command inserts the character with the specified character code. If you specify less than three octal digits, you should insert a space after the last digit. If given a numeric argument, this command inserts the octal character the specified number of times.

Otherwise, this command manipulates various quantities:

- The first character following CTRL-Q invokes one of the following operations:

B	backward	S	save
C	copy	T	transpose
D	delete	U	uppercase
F	forward	Z	reverse
L	lowercase	@	mark region
R	rubout		

- The second character following CTRL-Q invokes one of the following quantity types:

A	atom	S	sentence
C	character	W	word
D	definition	(list
H	buffer)	list
L	line	-	symbolic expression
P	paragraph	CLEAR SCREEN	page

You can also use numeric arguments with this command.

**Mode and Buffer
Attribute
Commands**

3.14 Zmacs contains major and minor modes. A *major mode* is the overall framework for the way you operate on the file. You always have one and only one major mode active for a buffer. If you are editing Lisp code, Common Lisp mode or Zetalisp mode is appropriate. Text mode is appropriate for text files.

A *minor mode* is used in conjunction with a major mode. It also changes the way Zmacs operates, but on a smaller scale than a major mode. You can have any minor mode with any major mode. You can also have more than one minor mode active at a time.

In addition to modes, Zmacs allows you to set other attributes that affect the buffer. You can set the fonts, the package, the base, and many others.

Section 2, Zmacs Operations, describes how to use modes and set other buffer attributes.

**Major Mode
Commands**

3.14.1 Zmacs contains the following major modes:

- Common Lisp — Designed for editing Common Lisp code.
- Zetalisp — Designed for editing Zetalisp code.
- Text — Designed for editing text.
- Fundamental — Designed for dealing with user-defined file types.
- Macsyma — Designed for editing Macsyma code.
- Ztop — Designed for editing within a read-eval-print loop.

Common Lisp Mode

Command

Keystroke: META-X Common Lisp Mode

Activates a major mode that sets up the editor for editing Common Lisp code. This command puts the Indent for Lisp command on the TAB key.

Zetalisp Mode

Command

Keystroke: META-X Zetalisp Mode

Activates a major mode that sets up the editor for editing Zetalisp code. This command puts the Indent for Lisp command on the TAB key.

Text Mode

Command

Keystroke: META-X Text Mode

Activates a major mode that sets up the editor for editing English text. This command puts the Tab to Tab Stop command on the TAB key.

Fundamental Mode Command

Keystroke: META-X Fundamental Mode

Activates a major mode that is Zmacs' fundamental mode. This mode is used for dealing with user-defined file types.

Macsyma Mode Command

Keystroke: META-X Macsyma Mode

Activates a major mode that sets up the editor for editing Macsyma code. This command modifies the delimiter dispatch tables appropriately for Macsyma syntax and makes the characters `/*` and `*/` comment delimiters. This command also puts the Indent Nested command on the TAB key.

Ztop Mode Command

Keystroke: META-X Ztop Mode

Activates a major mode that simulates a read-eval-print loop within a Zmacs buffer. This mode enables you to have a Lisp Listener inside the edit buffer. You can then save the buffer to a file. This mode supports most commands you would expect to use in the Lisp Listener.

**Minor Mode
Commands**

3.14.2 Zmacs contains the following minor modes:

- Any Bracket — Zmacs treats bracket characters such as `[]` and `{ }` like parentheses.
- Atom Word — All word commands act on Lisp atoms.
- Auto Fill — Zmacs automatically fills text.
- Electric Font Lock — Zmacs puts text in one font, comments in a second font, documentation strings in a third font, other strings in a fourth font, and function specifications in a fifth font.
- Electric Shift Lock — Zmacs uppercases everything except comments and strings.
- Overwrite — Typing over existing characters replaces them.
- RETURN Indents — The RETURN key indents a new line; the LINE FEED key does not.
- Uppercase Global Functions — Function names that are contained in the GLOBAL package are automatically uppercased.
- Word Abbreviation — Zmacs expands word abbreviations.

Any Bracket Mode Command

Keystroke: META-X Any Bracket Mode

Activates a minor mode in which the editor treats bracket characters such as [] and { } like parentheses. This treatment makes them easier to balance and so on. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Atom Word Mode Command

Keystroke: META-X Atom Word Mode

Activates a minor mode in which all word commands act on Lisp atoms. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Auto Fill Mode Command

Keystroke: META-X Auto Fill Mode

Activates a minor mode in which text is filled. When the right margin is reached, text is automatically put on the next line. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Electric Font Lock Mode Command

Keystroke: META-X Electric Font Lock Mode

Activates a minor mode to put text in font A, comments in font B, and documentation strings in font C. (A is the first font listed in the attribute list, B the second, and C the third.) If you supply font D, quoted strings that are not documentation for definitions (fourth in the top-level form) are put in font D; otherwise, they are put in font A. If you supply font E, all function specifications (second in the top-level form) are put in font E. Otherwise, they are put in font A. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Electric Shift Lock Mode Command

Keystroke: META-X Electric Shift Lock Mode

Activates a minor mode to uppercase everything except comments and strings. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Overwrite Mode Command

Keystroke: META-X Overwrite Mode

Activates a minor mode in which normal typing replaces existing text. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

RETURN Indents Mode Command

Keystroke: META-X Return Indents Mode

Activates a minor mode in which the RETURN key indents a new line and the LINE FEED key does not. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Word Abbrev Mode Command

Keystroke: META-X Word Abbrev Mode

Activates a minor mode for expanding word abbreviations. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Also refer to paragraph 3.5.4, Word Abbreviation Commands.

Uppercase Global Functions Mode Command

Keystroke: META-X Uppercase Global Functions Mode

Activates a minor mode in which function names that are contained in the GLOBAL package are automatically uppercased. When you type any word abbreviation mode delimiter except a hyphen (-), the editor checks the preceding atom to see if it needs uppercasing. (A word abbreviation mode delimiter is a space or any character that is not a letter or number.) A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

The two commands that follow are associated with the Uppercase Global Functions mode.

Uppercase Previous Global Function Command

Keystroke: META-X Uppercase Previous Global Function

Uppercases the atom under point if it is the name of a function in the GLOBAL package.

Uppercase Global Functions Region

Command

Keystroke: META-X Uppercase Global Functions Region

Uppercases the global functions in the region. If there is no region, this command uses the current definition (such as **defun**). With a numeric argument, this command lowercases all nonglobal function names in the region.

**Buffer Attribute
Commands**

3.14.3 The Buffer Attribute commands allow you to set attributes such as fonts and package for the buffer. They also allow you to reparse or update the attribute list in the top line of the buffer. Many files contain an attribute list. When you bring the file into a buffer, Zmacs uses the attributes defined in the attribute list for the buffer.

If you edit the attribute list (for example, add a new font to the font list), you need to *reparse* the attribute list for the new information to take effect. (Section 2, Zmacs Operations, describes how to edit the attribute list.)

If you perform an operation such as changing modes with a mode command, the change lasts only for that edit session unless you *update* the attribute list. By updating the attribute list, you make the changes permanent. (When you save the buffer to a file, the new attribute list is saved.) When you bring the file back into a buffer, Zmacs uses the attributes contained in the new attribute list.

You can set the following attributes for a buffer:

- Major mode — Tells Zmacs which major mode to use for the buffer.
- Package — Specifies the package to associate with the buffer.
- Fonts — Specifies which fonts to use in the buffer.
- Backspace — Specifies that backspace characters are to overprint on display.
- Base — Sets the base for numbers in the buffer (for example, 10 for decimal and 8 for octal).
- Lowercase — Specifies that the buffer contains lowercase or mixed-case data.
- Nofill — Specifies not to use Auto Fill mode without an explicit user command.
- Patch file — Specifies that the buffer is a patch file. This attribute allows you to redefine functions (and so on) that are already defined in other files without receiving warnings when the file is compiled or loaded.
- Tab width — Sets the displayed width of tab characters for the buffer.

Reparse Attribute List Command

Keystroke: META-X Reparse Attribute List

Reads the attribute list (-*-) again and adjusts the buffer characteristics accordingly. This command is used when you edit the attribute list and want the new attribute information to take effect. This command is particularly needed when you create a new buffer and type the attribute list yourself.

Update Attribute List Command

Keystroke: META-X Update Attribute List

Updates the attribute list (-*-) in the text from the current settings of the buffer. You use this command when you perform an operation such as changing modes with a mode command and you want to record that change in the attribute list. This update attribute list operation is the inverse of the reparse attribute list operation.

Set Backspace Command

Keystroke: META-X Set Backspace

Changes the backspace attribute of this buffer. A non-nil value causes backspace characters to actually overprint on display. You are prompted in the minibuffer for the new value of this attribute. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well.

Set Base Command

Keystroke: META-X Set Base

Changes the base for numbers appearing in this buffer (for example, 10 for decimal and 8 for octal). The change applies only to this buffer and overrides what the attribute list says. The command asks you whether to change the attribute list (-*-) in the text as well. If you supply a numeric argument with this command, it is used as the new value for the base. This entry is interpreted as a number in the current base unless followed by a decimal point (period) to indicate base 10. Otherwise, you are prompted in the minibuffer for the new value. The default is the value of the Emacs user variable `Default Base`.

NOTE: If you change the base with this command and do not change the attribute list to put a base attribute there, the new base is written into the attribute list when you save the file. If the attribute list contains a base attribute and you do not change it when executing this command, the old base stays in the attribute list when you save the file.

Set Fonts

Command

Keystroke: META-X Set Fonts

Changes the set of fonts to use. This command sets the fonts associated with the current buffer. You are prompted in the minibuffer for a list of font names, separated by spaces. You can specify up to 26 different fonts for a buffer. The command asks you whether to change the attribute list (-*-) in the text as well.

Also refer to paragraph 3.9, Font Commands.

Set Lowercase

Command

Keystroke: META-X Set Lowercase

Changes the lowercase attribute of this buffer. A non-nil value identifies the buffer as containing lowercase or mixed-case data. Therefore, Electric Shift Lock mode is not used as a default. You are prompted in the minibuffer for the new value of this attribute. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well.

Set Nofill

Command

Keystroke: META-X Set Nofill

Sets the nofill attribute of this buffer. A non-nil value prevents the use of Auto Fill mode without an explicit user command. You are prompted in the minibuffer for the new value of this attribute. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well.

Set Package

Command

Keystroke: META-X Set Package

Changes the package associated with the buffer or file. You are prompted in the minibuffer for a new package to be used when interned symbols that are read from this buffer (for example, when evaluating or compiling parts of the buffer). To specify a package that does not already exist, you must exit with CTRL-RETURN or press RETURN twice. Then you must confirm with yes.

After you specify the package, you are asked whether to change the attribute list (-*-) in the text. If you answer yes, the buffer's attribute list is modified to indicate that the buffer belongs in the new package. This modification affects all operations on the file once you save the buffer. If you answer no to the prompt, the change in package names lasts only for that editing session.

Then you are asked whether to resectionize the buffer. If you answer yes, all the function definitions in the buffer are recorded under symbols in the new package.

Also refer to paragraph 3.11.5, Package Commands.

Set Patch File

Command

Keystroke: META-X Set Patch File

Sets the patch file attribute of this buffer. This attribute allows you to redefine functions (and so on) that are already defined in other files without receiving warnings when the file is compiled or loaded.

A non-nil value identifies this buffer as a patch file. You are prompted in the minibuffer for the new value of this attribute. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well.

Set Tab Width

Command

Keystroke: META-X Set Tab Width

Sets the displayed width of tab characters for this buffer. This width is the separation of tab stops, measured in space characters. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well. If you supply a numeric argument with this command, it is used as the new value for the tab width. Otherwise, you are prompted in the minibuffer for the new value of this attribute. The default is 8.

Also refer to paragraph 3.19.6, Tab and Indentation Commands.

Set VSP

Command

Keystroke: META-X Set VSP

Sets the VSP (vertical interline spacing) for this buffer. The VSP is the number of blank rows of pixels between lines of text. The change applies only to this buffer and overrides what the attribute list specifies. The command asks you whether to change the attribute list (-*-) in the text as well. If you supply a numeric argument with this command, it is used as the new value for the VSP. Otherwise, you are prompted in the minibuffer for the new value of this attribute. The default is 2.

Mouse Commands 3.15 The Mouse commands allow the mouse to perform a variety of commands. You use mouse clicks to execute these commands instead of typing the names of the commands after pressing META-X. When the situations arise for these commands, the mouse documentation window tells you what the mouse can do.

NOTE: Many of the Mouse commands require that you double click a mouse button. If you double click too fast, the system sees only one click. If you double click too slow, the system sees two single clicks. You can use an alternative method to prevent misinterpretation: press and hold the CTRL key while you click the specified button.

Kill Yank Command

How to execute: double click middle

This command performs different operations depending on what the previous action was:

1. If there is a region marked, this command saves it on the kill history.
2. If the previous action was the one stated in number 1, this command deletes the region.
3. If the previous action was the one stated in number 2, this command yanks the top entry in the kill history.
4. If the previous action was the one stated in number 3, this command rotates backward through the kill history, performing yanks to replace the last yank.

Also refer to paragraph 3.6.2, Kill Commands, and paragraph 3.6.5, Yank Commands.

Mark Item Command

How to execute: click middle

Marks the item to which you are pointing with the mouse cursor.

Also refer to the paragraph 3.6.3, Mark Commands.

Mark Region Command

How to execute: hold down left mouse button

Moves point and mark to the position of the mouse cursor. Then, as you move the mouse cursor with the left button held down, point follows the mouse cursor.

Also refer to paragraph 3.6.3, Mark Commands.

Move Point Command

How to execute: click left

Moves point to the mouse cursor.

Also refer to paragraph 3.4, Cursor Movement Commands.

Move to Point Command

How to execute: double click left

Moves the mouse cursor to point.

Also refer to paragraph 3.4, Cursor Movement Commands.

Other Window Command

How to execute: position the mouse cursor on the other window and click left

Moves to the other window.

Also refer to paragraph 3.20, Window Commands.

Show Editor Menu Command

How to execute: click right

Shows the Editor menu.

Show System Menu Command

How to execute: double click right

Shows the System menu.

**Print
Commands**

3.16 The Print commands allow you to perform the following operations:

- Print a file, buffer, or region on a hard-copy device.
- Grind (pretty print) a function definition or the evaluation of a form. *Grind* means to indent Lisp code correctly (or to pretty print it). *Grind* works on a whole Lisp expression, whereas *indent* usually works on one or more lines that you specify or take as the default.

In addition to the following commands, you can print a buffer from the List Buffers and the Edit Buffers menus (refer to paragraph 3.2.4, List and Edit Buffer Commands). You can print a file from the Dired listing (refer to paragraph 3.7.2, Edit a Directory (Dired) Commands).

Grind Definition Command

Keystroke: META-X Grind Definition

Grinds the definition of a function into the buffer. You are prompted in the minibuffer for the name of the function. This command inserts the properly indented definition at point.

Also refer to paragraph 3.12.2, Grind (Pretty Print) Commands.

Grind Expression Command

Keystroke: META-X Grind Expression

Grinds the evaluation of a form into the buffer. You type a form into the minibuffer. This command evaluates it and inserts the properly indented result at point.

Also refer to paragraph 3.12.2, Grind (Pretty Print) Commands.

Print All Buffers Command

Keystroke: META-X Print All Buffers

Query prints all buffers on the default hard-copy device (provided that there is one). You are asked about each buffer individually.

Also refer to paragraph 3.2.7, Print Buffer Commands.

Print Buffer Command

Keystroke: META-X Print Buffer

Prints a buffer on the default hard-copy device (provided that there is one). You are prompted in the minibuffer for the name of the buffer. The default is the current buffer. You can press CTRL-SHIFT-F to specify a filename.

Also refer to paragraph 3.2.7, Print Buffer Commands.

Print File Command

Keystroke: META-X Print File

Prints a specified file on the default hard-copy device (provided that there is one). You are prompted in the minibuffer for the name of the file.

Also refer to paragraph 3.8.6, Print File Commands.

Print Region Command

Keystroke: META-X Print Region

Prints the region on the default hard-copy device.

Quick Print Buffer Command

Keystroke: META-SHIFT-P

Prints the current buffer on the default hard-copy device.

Also refer to paragraph 3.2.7, Print Buffer Commands.

Scroll Commands

3.17 The Scroll commands allow you to scroll the editor buffer window. The following list summarizes the operations you can perform with the Scroll commands:

- Scroll forward or backward one screen of text. With a numeric argument, scroll by that number of *lines*.
- Scroll forward or backward several screens of text.
- Scroll to the beginning or the end of the buffer.
- Scroll to the next or previous page. A *page* is delimited by the page character, which is created by pressing CTRL-Q CLEAR SCREEN.
- Scroll the screen to position all of the current definition in the window.
- Scroll the other window forward or backward several lines.

Goto Beginning

Command

Keystroke: META-< or HYPER-↑

Goes to the beginning of the buffer. With an argument from 0 to 10, this command goes forward that many tenths of the length of the buffer, starting from the beginning.

Also refer to paragraph 3.4, Cursor Movement Commands.

Goto End

Command

Keystroke: META-> or HYPER-↓

Goes to the end of the buffer. With an argument from 0 to 10, this command goes backward that many tenths of the length of the buffer, starting from the end.

Also refer to paragraph 3.4, Cursor Movement Commands.

Next Page

Command

Keystroke: CTRL-X]

Scrolls to the next page. A page is delimited by the page character, which is created by pressing CTRL-Q CLEAR SCREEN.

Next Screen

Command

Keystroke: CTRL-V or CTRL-↓

Scrolls forward one screen of text. With an argument of *n*, this command scrolls the window forward *n* lines (not screens).

Next Several Screens Command

Keystroke: META-X Next Several Screens

Scrolls forward several screens of text. A numeric argument specifies the number of screens to scroll.

Previous Page Command

Keystroke: CTRL-X [

Scrolls to the previous page. A page is delimited by the page character, which is created by pressing CTRL-Q CLEAR SCREEN.

Previous Screen Command

Keystroke: META-V or CTRL-↑

Scrolls backward one screen of text. With an argument of *n*, this command scrolls the window backward *n* lines (not screens).

Previous Several Screens Command

Keystroke: META-X Previous Several Screens

Scrolls backward several screens of text. A numeric argument specifies the number of screens to scroll.

Reposition Window Command

Keystroke: META-CTRL-R

Attempts to display all of the current definition in the window. The definition can be any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defun**, **defflower**, **defvar**, and so on.

- If the beginning of the definition is on the screen, this command scrolls forward to bring the end onto the screen.
- If the beginning of the definition is off the screen, this command scrolls backward so that the beginning appears, but it does not push point off the bottom of the screen.
- If the beginning of the definition is at the top of the screen, this command tries omitting or including the comments before it.
- If the definition is entirely on the screen, this command positions it at the top of the screen. (With a numeric argument, the definition is positioned at the bottom of the screen.)

Also refer to paragraph 3.20, Window Commands.

Scroll Other Window

Command

Keystroke: META-CTRL-V

Scrolls the other window forward several lines when two windows are exposed. A numeric argument specifies the number of lines. A negative argument scrolls backward. The default is one screen forward.

Also refer to paragraph 3.20, Window Commands.

Search and Replace Commands

- 3.18 Zmacs provides extensive search and replace capabilities:
- Searching — You can search for simple characters or complicated combinations of strings (such as two words at the same time).
 - Replacing — You can replace strings globally or by responding to queries. You can also replace two strings at the same time.
 - Tag tables — You can specify a group of files through which you want to search and replace.

Search Commands

3.18.1 Zmacs provides three different types of searches:

- String search finds the string that you specify.
- Incremental search finds intermediate strings while you are typing. For example, if you want to find DEFMINOR, it finds the first D when you type it. When you type E, it finds the first DE and so on.
- Extended search lets you make complicated searches. You can search for combinations of strings by using special search characters. For example, you can search for two words at the same time.

All of these types of searches allow you to search forward or backward. These also allow you to switch directions (reverse) in the middle of the search.

In addition to these three types of searches, commands are available that let you perform searches for a pattern or for text lines containing a given string.

String Search Commands

3.18.1.1 A string search finds the string that you specify. You can search forward or backward for the string. When you type a string and press RETURN, the string search operation searches for the string. It lets you type the whole string before it starts looking.

If you press HELP when you are performing a string search, you receive help about string search. You can press CTRL-D to retrieve strings that you have searched for previously. They are stored on the *search ring*. The search ring stores only three strings, but you can change the Zmacs user variable Search Ring Max to make it store more than three previous strings.

Pressing CTRL-S ESCAPE when you are performing an incremental search also invokes the string search.

String Search

Command

Keystroke: META-X String Search or CTRL-S ESCAPE

Reads a string and searches for it. While you are typing the search string, the keystrokes in the following table have special meaning. You can display this list of keystrokes by entering META-X String Search and then pressing HELP. Do not press RETURN.

Keystroke	Meaning
CTRL-B	Searches forward from the beginning of the buffer.
CTRL-E	Searches backward from the end of the buffer.
CTRL-D	Retrieves a string to search for from the search ring of previously searched strings.
CTRL-F	Leaves point at the top of the window, if the window must be recentered.
CTRL-G	Aborts the search.
CTRL-L	Redisplays the type-in line (where the search string is entered).
CTRL-Q	Quotes the next character so that you can search for a special character. For example, you can press CTRL-Q RETURN to search for the beginning of a line. Basically, CTRL-Q unconditionally inserts the next key into the buffer, even if the key normally does something else.
CTRL-R	Reverses the direction of the search.
CTRL-S	Performs the search and returns to the command loop (to accept another search string).
CTRL-U	Deletes all characters typed so far.
CTRL-V	Delimited search: finds only occurrences of the string surrounded by delimiters (defined by the major mode).
CTRL-W	Word search: searches for words in sequence, regardless of intervening punctuation, whitespace, new lines, and other delimiters.
CTRL-Y	Appends the string on top of the kill history to the search string.
RUBOUT	Rubs out the previously typed character.
CLEAR INPUT	Deletes all characters typed so far.
ESCAPE	Performs the search and exits.

If you search for the empty string, the default is used. The default is the last string you searched for. (If you pressed ABORT to quit searching for the last string, the string is not remembered as the default, even if you already found several occurrences of it.)

If you do not search for the empty string, the string you type becomes the default string, and the default is saved on the search ring unless it is a single character.

Reverse String Search

Command

Keystroke: META-X Reverse String Search or CTRL-R ESCAPE

Reads a string and searches backward for it. Refer to the previous description of the String Search command for a list of the options you can also use for this command. You can display this list of options by entering META-X Reverse String Search and then pressing HELP. Do not press RETURN. You must press ESCAPE to begin the search.

Incremental Search Commands

3.18.1.2 An incremental search finds strings as you type them. The string is accumulated, and the incremental search operation searches for what you have typed so far. You can search forward or backward. CTRL-S invokes the Incremental Search command. CTRL-R invokes the Reverse Incremental Search command. For example, if you are looking for CATS, it finds the first C when you type it. Then it finds CA and so on. Once you find CATS, you can press CTRL-S again to find the next occurrence.

Pressing HELP when you are performing an incremental search provides help about the incremental search.

Incremental Search

Command

Keystroke: CTRL-S

Searches for a character string. As characters are typed, the accumulated string is displayed and searched for. The keystrokes in the following table have special meaning:

Keystroke	Meaning
RUBOUT	Removes characters in the accumulated string. This keystroke also applies to secondary incremental searches. A secondary search occurs when you press CTRL-R to find the next occurrence of whatever string you are looking for. RUBOUT erases a character and searches forward for the string. For example, it erases the S from the string CATS and finds CAT.
ESCAPE	Stops the search and leaves you where the cursor is. However, if the search string is empty, ESCAPE invokes the String Search command.
CTRL-Q	Quotes characters so that you can search for special characters. For example, you can press CTRL-Q RETURN to search for the beginning of a line. Basically, CTRL-Q unconditionally inserts the next key into the buffer, even if the key normally does something else.

Keystroke	Meaning
CTRL-S	Allows you to repeat the search with the same string. RUBOUT erases a character from the string and searches backward for the string. If CTRL-S or CTRL-R is the first character pressed after a CTRL-S, the previous search string is used again.
CTRL-R	Allows you to search backwards. If CTRL-R or CTRL-S is the first character pressed after a CTRL-R, the previous search string is used again.
ABORT	Takes you back to where the search started. Most of the CTRL or META characters also take you out of the incremental search. Generally, they leave you where the cursor is.

Reverse Incremental Search Command

Keystroke: CTRL-R

Searches backwards for a character string. As characters are typed, the accumulated string is displayed and searched for. The keystrokes in the following table have special meaning:

Keystroke	Meaning
RUBOUT	Removes characters in the accumulated string. This keystroke also applies to secondary incremental searches. A secondary search occurs when you press CTRL-S to find the next occurrence of whatever string you are looking for. RUBOUT erases a character and searches backward for the string. For example, it erases the S from the string CATS and finds CAT.
ESCAPE	Stops the search and leaves you where the cursor is. However, if the search string is empty, ESCAPE invokes the String Search command.
CTRL-Q	Quotes characters so that you can search for special characters. For example, you can press CTRL-Q RETURN to search for the end of a line. Basically, CTRL-Q unconditionally inserts the next key into the buffer, even if the key normally does something else.
CTRL-R	Allows you to repeat the search with the same string. RUBOUT erases a character from the string and searches forward for the string. If CTRL-R or CTRL-S is the first character pressed after a CTRL-R, the previous search string is used again.

Keystroke	Meaning
CTRL-S	Allows you to search forward. If CTRL-S or CTRL-R is the first character pressed after a CTRL-S, the previous search string is used again.
ABORT	Takes you back to where the search started. Most of the CTRL or META characters also take you out of the incremental search. Generally, they leave you where the cursor is.

Yank Search String	Command
---------------------------	----------------

Keystroke: CTRL-S CTRL-S

Inserts the last incremental search string into the minibuffer.

Also refer to paragraph 3.6.5, Yank Commands.

Incremental Search Example Suppose you want to search for the PAGE character (**<PAGE>**). You press CTRL-S, then CTRL-Q to give you one of the special characters. Then you press the CLEAR SCREEN key, which gives you the PAGE character.

If you want to stay there, you press ESCAPE. Notice that the minibuffer says *Point Pushed*. This message means that your previous location has been added to the point PDL. If you press META-STATUS, that location is the first entry on the list.

Retrieving a Previous String You can retrieve a previous search string. Suppose you type a string, find it, and then press ESCAPE because you want to stop your search. If you want to search for the same string again, you press CTRL-S twice to retrieve the string that you searched for previously. Then you can look for the next occurrence of the string. You do not have to retype the string. Likewise, pressing CTRL-R twice retrieves the string so that you can reverse direction.

*Extended Search
Commands*

3.18.1.3 Extended search commands allow you to make very complicated searches either forward or backward. You can search for combinations of strings by using special search characters, which are invoked by CTRL-H.

NOTE: The extended search characters work on other commands besides Extended String Search. The message *Extended Search characters* appears on the right side of the mode line. For example, the characters work on the *Apropos*, *Variable Apropos*, and *Function Apropos* commands. (Refer to the Help commands described in paragraph 3.10, *Help, Documentation, and Undo Commands*.)

Extended String Search Command

Keystroke: META-X Extended String Search

Searches for a complicated string. You are prompted in the minibuffer for the string. CTRL-H is a prefix for special search characters.

Extended Reverse String Search Command

Keystroke: META-X Extended Reverse String Search

Searches backward for a complicated string. You are prompted in the minibuffer for the string. CTRL-H is a prefix for special search characters.

The extended search characters allow you to perform the following operations. Each special search character appears in a lozenge (that is, a hexagon) when you enter it by using one of the keystrokes. The search characters are standard characters that you can see in the search font.

Search Range Direction Options These options control forward or backward search from point. The default is to search forward from point. The search range direction options are shown in the following table:

Keystroke	Description
CTRL-H CTRL-B	Searches from the beginning of the buffer. Pressing CTRL-H CTRL-B again searches from point.
CTRL-H CTRL-E	Searches from the end of the buffer. Pressing CTRL-H CTRL-E again searches from point.
CTRL-H CTRL-R	Reverses the direction of the search.

Wildcard Character Options These options allow you to create a search pattern where certain positions require a specific character and other positions can be wildcards for any of a particular set of characters. The following table shows the wildcard character options:

Keystroke	Description
CTRL-H CTRL-X	Searches for a string with any character in this position.
CTRL-H A	Searches for a string with any alphabetic character in this position.
CTRL-H -	Searches for a string with any delimiter character in this position.
CTRL-H space bar	Searches for a string with any whitespace character in this position.
CTRL-H ~	Searches for a string that does <i>not</i> have the next item in this position.

Repeated Wildcard Character Option This option allows you to create a search pattern where certain positions require a specific character and other positions can be wildcards that specify zero or more occurrences of the next character. For example, you may search for two words separated by one or more blanks.

CTRL-H* is the keystroke for this option. It searches for a string with zero or more occurrences of the next character in this position.

Multiple Condition Options These options allow you to specify *and* and *or* connectives and to group multiple strings so that you can tell which one is executed first.

Keystroke	Description
CTRL-H (CTRL-H)	Uses (and) to control logical combinations of search patterns.
CTRL-H CTRL-O	Searches for either the previous string <i>or</i> the next string. Use CTRL-H (and CTRL-H) to control logical combinations of search patterns, if needed.
CTRL-H &	Searches for a line with both the previous <i>and</i> the next string. Use CTRL-H (and CTRL-H) to control logical combinations of search patterns, if needed.

String Display Option This option tells where to display the string.

CTRL-H CTRL-F moves the cursor to the next string found; CTRL-F causes it to be positioned on the top line.

Abort Option This option allows you to abort when entering a wildcard option.

CTRL-H ABORT or any other key not defined in the list of CTRL-H keystrokes returns you to the search string specification that you are entering in the minibuffer.

As an example of using these options, suppose you want to search for either the name *Christa* or the names *Mary Christa* followed by the name *McNulty*. The following procedure shows how you specify this search:

1. Press CTRL-H (.
2. Type *christa*.
3. Press CTRL-H CTRL-O.
4. Type *Mary christa*.
5. Press CTRL-H).
6. Type *McNulty*.

The following line shows how the screen looks:

```
<( > Christa <OR> Mary Christa <)> McNulty
```

*Miscellaneous
Search Commands*

3.18.1.4 These commands allow you to perform the following operations:

- Find a pattern.
- List text lines that contain a given string.
- List symbols satisfying the given predicate. You specify a function of one symbol. All symbols are found that cause the function to return `t` when they are passed to the function as its argument.
- List the packages that contain a specified symbol.

Find Pattern

Command

Keystroke: META-X Find Pattern

Moves to the next occurrence of the given pattern of Lisp code. In matching, differences in whitespace characters are ignored except for characters that are quoted or inside strings.

- You can use the character `?` as an atom in the pattern to match any symbolic expression in the buffer (that is, *one* symbolic expression).
- You can use the characters `??` as an atom in the pattern to match any number of symbolic expressions (that is, *zero or more* symbolic expressions).

Patterns starting with infrequent characters, such as opening parentheses, are found more quickly. Those patterns starting with common letters are likely to be found more slowly.

This command puts the cursor at the front of the matching pattern. With a negative argument, this command searches backward. With an empty pattern string, this command repeats the last pattern specified.

Suppose you have the following patterns of code in your buffer:

1. (a b d e)
2. (a b g d e)
3. (a b c d e)
4. (a b c d e f)

The following table provides examples of using the characters `?` and `??` when searching for these patterns and shows which patterns the examples find:

Search Pattern	Patterns Found
(a b ? d e)	2 and 3
(a b ?? d e)	1, 2, and 3
(a b ?? e)	1, 2, and 3
(a b c ??)	3 and 4
(a b ? d e ??)	4

Lisp Match Search

Command

Keystroke: CTRL-SHIFT-S

Moves to the next occurrence of the given pattern of Lisp code. In matching, differences in whitespace characters are ignored except for characters that are quoted or inside strings.

- You can use the character `?` as an atom in the pattern to match any symbolic expression in the buffer (that is, *one* symbolic expression).
- You can use the characters `??` as an atom in the pattern to match any number of symbolic expressions (that is, *zero or more* symbolic expressions).

Patterns starting with infrequent characters, such as opening parentheses, are found more quickly. Those patterns starting with common letters are likely to be found more slowly.

This command puts the cursor after the matching pattern. With a negative argument, this command searches backward. With an empty pattern string, this command repeats the last pattern specified.

Suppose you have the following patterns of code in your buffer:

1. (a b d e)
2. (a b g d e)
3. (a b c d e)
4. (a b c d e f)

The following table provides examples of using the characters `?` and `??` when searching for these patterns and shows which patterns the examples find:

Search Pattern	Patterns Found
(a b ? d e)	2 and 3
(a b ?? d e)	1, 2, and 3
(a b ?? e)	1, 2, and 3
(a b c ??)	3 and 4
(a b ? d e ??)	2, 3, and 4

List Matching Lines

Command

Keystroke: META-X List Matching Lines

Displays text lines that contain a given string. The display is mouse-sensitive. If you select a line with the mouse, point moves to the line.

With an argument, this command shows the next *n* lines containing the string, where *n* is the numeric argument. If you do not supply an argument, this command displays all lines containing the string.

List Matching Symbols

Command

Keystroke: META-X List Matching Symbols

Lists symbols satisfying the given predicate. You specify a function of one symbol. This command finds all symbols that cause the function to return t when they are passed to the function as its argument. The following rules apply:

- Without a CTRL-U argument, this command searches the current package.
- With one CTRL-U, this command searches all packages.
- With two CTRL-Us, this command asks for a package.

Occur

Command

Keystroke: META-X Occur

Lists all occurrences in the file of the string you specify. The display is mouse-sensitive.

Where Is Symbol

Command

Keystroke: META-X Where Is Symbol

Shows which packages contain the specified symbol.

Also refer to the Help commands described in paragraph 3.10, Help, Documentation, and Undo Commands.

Replace Commands 3.18.2 The Replace commands allow you to replace the following items:

- A string (globally or with query)
- Two sets of strings at the same time (multiple query)
- Two strings with one another at the same time (exchange)
- Atoms

You can also delete all lines that contain a specified string or delete all lines that do not contain a specified string.

You can replace a string globally or with queries. To replace a string globally, you use the Replace String command (CTRL-%). Once you specify the replacement string for the original string, it replaces every occurrence of the original string.

To replace a string with queries, you use the Query Replace command (META-%). This command provides you with the following options:

- Finds the string to replace and asks you whether to replace it.
- Skips to the next occurrence.
- Replaces the string and shows the result.
- Replaces the string and exits.
- Exits without replacing.
- Returns to the previous occurrence of the string.
- Kills the string and enters recursive edit.
- Enters edit mode recursively.
- Redisplays.
- Replaces all remaining strings without asking.

You can also perform multiple query replaces with some of the other commands.

As an example of Query Replace, suppose you name a variable `x` and then realize that the name is not very descriptive. You can use Query Replace to go through your file and replace every occurrence of the name `x`. The command asks you if you want to replace at each occurrence. If you do not want it to continue asking you at each occurrence, you can type an exclamation point (!) to have the remaining occurrences replaced automatically.

Atom Query Replace Command

Keystroke: META-X Atom Query Replace

Performs a query replace, replacing only delimited atoms rather than strings.

Delete Matching Lines Command

Keystroke: META-X Delete Matching Lines

Deletes all lines containing the specified string. This command covers all text from point to the end of the buffer.

Delete Non Matching Lines Command

Keystroke: META-X Delete Non Matching Lines

Deletes all lines not containing the specified string. This command covers all text from point to the end of the buffer.

Multiple Query Replace

Command

Keystroke: META-X Multiple Query Replace

Performs a query replace on two sets of strings at the same time. You are prompted in alternate minibuffers for the strings. Pressing RETURN without entering anything ends the prompting. With a positive argument, strings must be surrounded by delimited characters (such as whitespace, punctuation, and so on). With a negative argument, delimited atoms are replaced, rather than words.

Refer to the documentation of the Query Replace command later in this section for more information.

Multiple Query Replace From Buffer

Command

Keystroke: META-X Multiple Query Replace From Buffer

Performs a multiple query replace from the contents of the specified buffer. First, you set up a buffer that contains the strings you want replaced. You pair the string you want replaced with the replacement string on one line. The string you want replaced comes first. (You can place multiple pairs on the same line.) The following lines show an example:

```
old new
good bad
win loss
start finish
```

Next, you go to the buffer where you want to make the replacements and enter this command. You are prompted in the minibuffer for the buffer (or file) that contains the replacement strings. Then, the command searches the buffer and stops at each of the strings you want replaced. Refer to the documentation of the Query Replace command later in this section for information on the options you have available at this point.

Query Exchange

Command

Keystroke: META-X Query Exchange

Performs a query replace in which two strings are exchanged with one another at the same time. With a positive argument, strings must be surrounded by delimited characters (such as whitespace, punctuation, and so on). With a negative argument, delimited atoms are replaced, rather than words.

Query Replace

Command

Keystroke: META-%

Replaces a string, asking about each occurrence. You are prompted for each string. If you first specify *foo* and then *bar*, this command finds the first *foo*,

recenters the buffer if necessary, and waits for you to choose one of the options shown in the following table:

Keystroke	Description
Space bar	Replaces <i>foo</i> with <i>bar</i> and shows the next <i>foo</i> .
RUBOUT	Does not replace, but shows the next <i>foo</i> .
Comma	Replaces this <i>foo</i> and shows the result, waiting for a space bar, CTRL-R, or ESCAPE.
Period	Replaces this <i>foo</i> and exits.
ESCAPE	Exits.
^	Returns to the site of the previous <i>foo</i> (actually, pops the point PDL).
CTRL-W	Kills this <i>foo</i> and enters recursive edit. To exit the recursive edit, press the END key.
CTRL-R	Enters edit mode recursively. To exit the recursive edit, press the END key.
CTRL-L	Redisplays the screen.
!	Replaces all remaining <i>foos</i> without asking.

Any other character exits, but the character is inserted into the buffer. ESCAPE exits without inserting a character into the buffer.

If the Zmacs user variable Case Replace is *t*, *bar*'s initial letter is capitalized if *foo*'s initial letter is capitalized. In other words, the way *bar*'s capitalization appears in the buffer depends on *foo*'s capitalization in the buffer, not on how you type it.

If you supply a numeric argument, this command ignores *foos* bounded on both sides by delimiter characters.

Replace String

Command

Keystroke: CTRL-%

Replaces all occurrences of a given string with another string. You are prompted for two strings: to replace all *foos* with *bars*, type *foo*, press RETURN, type *bar*, and press RETURN. Without a numeric argument, the command replaces all occurrences of *foo* after point. With a numeric argument, the command replaces the specified number of occurrences of *foo*.

If the Zmacs user variable Case Replace is *t*, *bar*'s initial letter is capitalized if *foo*'s initial letter is capitalized. In other words, the way *bar*'s capitalization appears in the buffer depends on *foo*'s capitalization in the buffer, not on how you type it.

Tag Commands **3.18.3** The Tag commands allow you to specify a group of files through which you want to search and replace. The files are then treated as one file. You can also perform some compile and evaluate operations on the group of files.

You specify the group of files by creating a tag table. The following commands allow you to make a tag table:

- **Select System as Tag Table (META-X Select System as Tag Table)** — Asks you for any system name that is defined by `defsystem`. All files in the system are then used in the tag table.
- **Select All Buffers as Tag Table (META-X Select All Buffers as Tag Table)** — Selects as a tag table all files that are read into buffers.
- **Visit Tag Table (META-X Visit Tag Table)** — Reads in the specified tag table file.
- **Select Tag Table (META-X Select Tag Table)** — Allows you to select a system for a tag table (like the **Select System as Tag Table** command), select all buffers for a tag table (like the **Select All Buffers as Tag Table** command), select a tag table file (like the **Visit Tag Table** command), or select an existing tag table.

After you create the tag table, you can use the other Tag commands to perform search and replace operations. You can also compile, evaluate, list, and edit changed definitions in the tag table files. All of these commands cycle through the files in the order specified by the tag table.

List Tag Tables Command

Keystroke: META-X List Tag Tables

Lists the names of all the tag table files read in.

Current Tag Table Command

Keystroke: META-X Current Tag Table

Tells which tag table you are currently using.

Next File Command

Keystroke: META-X Next File

Moves to the next file in the tag table.

Select All Buffers as Tag Table Command

Keystroke: META-X Select All Buffers as Tag Table

Selects as a tag table all files currently read into buffers.

This command causes commands such as Tags Search, Tags Query Replace, and Tags Compile Changed Definitions to look through all the files currently being visited.

Select System as Tag Table Command

Keystroke: META-X Select System as Tag Table

Makes the files in a system behave like a tags file.

Select Tag Table Command

Keystroke: META-X Select Tag Table

Makes a tag table current for commands such as Tags Search. This command prompts you for one of the following options:

- F — The command expects the tag table to be a file in one of the two formats described in the documentation of the Visit Tag Table command later in this section.
- S — The command makes the files in a system (defined by `defsystem`) behave like a tag table.
- B — The command makes all buffers currently read into files behave like a tag table.

This command displays the E option if you execute it after you have selected a tag table. This option cause the command to select an existing tag table.

Tags Compile Changed Definitions Command

Keystroke: META-X Tags Compile Changed Definitions

Compiles any edited definitions in the tag table files. With a numeric argument, this command asks about each definition individually.

Also refer to paragraph 3.3, Compile and Evaluate Commands.

Tags Edit Changed Definitions Command

Keystroke: META-X Tags Edit Changed Definitions

Edits any definitions in the tag table that have been edited. A definition is changed if it has been modified since one of the following events:

- The file was read in (numeric argument 1 or no argument).
- The file was read in or saved (numeric argument 2).
- The definition was compiled (numeric argument 3).

Tags Evaluate Changed Definitions Command

Keystroke: META-X Tags Evaluate Changed Definitions

Evaluates any definitions in the tag table that have been edited. With a numeric argument, this command asks about each definition individually.

Also refer to paragraph 3.3, Compile and Evaluate Commands.

Tags List Changed Definitions Command

Keystroke: META-X Tags List Changed Definitions

Lists any definitions in the tag table that have been edited. A definition is changed if it has been modified since one of the following events:

- The file was read in (numeric argument 1 or no argument).
 - The file was read in or saved (numeric argument 2).
 - The definition was compiled (numeric argument 3).
-

Tags Multiple Query Replace Command

Keystroke: META-X Tags Multiple Query Replace

Performs a query replace within the tag table files.

Refer to the documentation on Multiple Query Replace in paragraph 3.18.2, Replace Commands, for more information.

Continue Tags Multiple Query Replace Command

Keystroke: META-X Continue Tags Multiple Query Replace

Continues the last tag's multiple query replace.

Tags Multiple Query Replace From Buffer Command

Keystroke: META-X Tags Multiple Query Replace From Buffer

Performs a multiple query replace from the contents of the specified buffer. First, you set up a buffer that contains the strings you want replaced. You pair the string you want replaced with the replacement string on one line. The string you want replaced comes first. (You can place multiple pairs on the same line.) The following lines show an example:

```
old new
good bad
win loss
start finish
```

Next, you activate a tag table that contains the files where you want to make the replacements, and you enter this command. You are prompted in the minibuffer for the buffer (or file) that contains the replacement strings. Then, this command searches the tag table and stops at each of the strings you want replaced. Refer to the documentation on the Query Replace command in paragraph 3.18.2, Replace Commands, for information on the options you have available at this point.

Tags Query Replace Command

Keystroke: META-X Tags Query Replace

Performs a query replace within all the files in the selected tag table, one file at a time.

Refer to the documentation on the Query Replace command in paragraph 3.18.2, Replace Commands, for more information.

Continue Tags Query Replace Command

Keystroke: META-X Continue Tags Query Replace

Continues the last tag's query replace.

Tags Search Command

Keystroke: META-X Tags Search

Searches for the specified string within the files of the tag table. String search, incremental search, and extended search are available.

Tags Search List Definitions Command

Keystroke: META-X Tags Search List Definitions

Lists the definitions of the files of the tag table containing the specified string. You are prompted in the minibuffer for the string (extended search characters are allowed). This command searches the files in the tag table, recording the definitions that contain the string as a list of possibilities to visit.

Tags Search Next Occurrence Command

Keystroke: META-X Tags Search Next Occurrence

Searches for the next occurrence of the search string.

Visit Tag Table

Command

Keystroke: META-X Visit Tag Table

Reads in the specified tag table file. This command goes through the tag table and marks the name of each tag as being a possible section (definition) of its file. Later, the Edit Definition command (META-.) sees these marks and determines which file to use. You are prompted in the minibuffer for the name of the file.

This command accepts two formats for the tag table file. The following list details the rules for the first format. This type of format is also produced by the commands that create a tag table for you, such as the Select System as Tag Table command.

- The file is in a directory, and it can reference only the files in that directory.
- A sequence of items appears in the file, and you can repeat the sequence as many times as you want. The following items describe the lines in the sequence:
 - The first line contains a filename and a version number (which is optional). The file is assumed to be in the same directory as the tag table file. You cannot specify the host and directory names.
 - The second line can start with anything you want but must end with a comma and a mode name. No spaces can appear between the comma and the end of the line. The command scans for the comma. The mode can be one of the following: Common Lisp, Zetalisp, or Text.
 - You can end the sequence on the next line or put additional items as follows:
 - (def followed by a space or tab and a symbol if you use Common Lisp or Zetalisp mode
 - .def followed by a space or tab and a symbol if you use Text mode

You can have as many lines as you want. Because the symbol is inserted into the data structure, completion is available for it. (The symbol is assumed to be a function name.) The Edit Definition command (META-.) looks for the function as if it were in the file you are describing in the sequence.

The following lines show an example of this sequence:

```
FOO.LISP
,Common-Lisp
(def bar
```

Completion is available on bar. If you can use META-. on bar, the command goes into the file FOO and tries to find bar.

- To end the sequence, move to the next line after (def and, in column 0, press SYMBOL-W, which produces the logical OR symbol.

- After you end one sequence, you can begin a sequence for a new file.
- No blank lines can be at the end of the file, and the logical OR symbol must end the file.

The following list describes the rules for the second format:

- The first line is a special identifier for this type of format, as follows:

`FILES IN user-specified-name`

This identifier contains the word `FILES`, a blank, the word `IN`, a blank, and then a name that you specify. The other format always contains a filename in the first line.

The *user-specified-name* is the name that you specify when executing the Select Tag Table command. The Current Tag Table command lists this name when the file is the current tag table. The List Tag Table command lists the name if it is one of the tag table files read in.

- On the next line, you specify the name of each file in the tag table. This format does not assume that the file is in the same directory. You can use directory names.
- Blank lines are acceptable.

You can use the Visit Tag Table command to read the buffer that is in the second format and make it into a tag table.

You save a tag table to disk by using the Write File command (`CTRL-X CTRL-W`).

**Text Format
Commands**

3.19 The Text Format commands provide a variety of operations for formatting text. The following summarizes these operations:

- Exchange commands — Allow you to exchange (transpose) characters, lines, regions, symbolic expressions, and words.
- Fill commands — Allow you to fill and adjust text. *Fill* means to put as much text as possible on the current line without exceeding the right margin. *Adjust* means to justify or add spaces until the right margins of each line align.
- Lowercase and Uppercase commands — Allow you to lowercase or uppercase regions of text, regions of Lisp code, or words.
- Miscellaneous Text Format commands — Provide several options for formatting text, such as inserting blank lines, setting the right margin, and setting a goal column.
- Reverse commands — Allow you to reverse the order of lines or the elements of a list.
- Sort commands — Allow you to sort regions by lines or paragraphs or by using keyboard macros.
- Tab and Indentation commands — Provide a variety of options to help you indent your Lisp programs.
- Just One Space command — Allows you to manipulate whitespace. *Whitespace* is any character contained in the Zmacs user variable `Whitespace Chars`.

**Exchange
Commands**

3.19.1 The Exchange (Transpose) commands allow you to interchange the following items with each other:

- Characters
- Lines
- Regions
- Symbolic expressions
- Words

Refer to the Various Quantities command (paragraph 3.4.5) for other Exchange commands, such as those that allow you to exchange sentences and paragraphs. You can press CTRL-Q T P to exchange the paragraph in which the cursor is positioned with the paragraph before it.

Zmacs also contains Reverse commands that allow you to reverse the following items:

- Elements of a list
- The order of lines

Exchange Characters

Command

Keystroke: CTRL-T

Interchanges the characters before and after point (that is, the character before the cursor and the one that the cursor is on), as follows:

- With a positive argument, this command interchanges the characters before and after point, moves right one character, and repeats this operation the specified number of times.
- With a negative argument, this command interchanges the two characters to the left of the cursor, moves between them, and repeats this operation the specified number of times, exactly undoing the operation this command performs with a positive argument.
- With an argument of 0, this command interchanges the characters at point and mark.

No argument produces the same result as an argument of 1, except that at the end of a line the previous two characters are interchanged.

Exchange Lines

Command

Keystroke: CTRL-X CTRL-T

Interchanges the line the cursor is on with the line before the cursor, as follows:

- With a positive argument, this command interchanges the line the cursor is on with the line before the cursor, moves down one line, and repeats this operation the specified number of times.
- With a negative argument, this command interchanges the line before the cursor with the line two lines before the cursor, moves up one line, and repeats this operation the specified number of times, exactly undoing the operation this command performs with a positive argument.
- With an argument of 0, this command interchanges the lines at point and mark.

Exchange Regions

Command

Keystroke: CTRL-X T

Exchanges two regions. This command uses point and mark (if mark is the first entry on the point PDL) and then the two previous points on the point PDL. For example, you can push two points onto the point PDL by using the Set Pop Mark command (CTRL-space bar) and then push a third point (which puts mark there) and use the keyboard (not the mouse) to move point to the end of the second region. Then, you execute this command to exchange the regions.

Alternatively, if the regions are next to each other, you can push point at the beginning of one region, between them, and at the end of the second region. Then, you execute this command to exchange the regions.

Exchange Sexps

Command

Keystroke: META-CTRL-T

Interchanges the symbolic expressions before and after point, as follows:

- With a positive argument, this command interchanges the symbolic expressions before and after point, moves right one symbolic expression, and repeats this operation the specified number of times.
- With a negative argument, this command interchanges the two symbolic expressions to the left of point, moves between them, and repeats this operation the specified number of times, exactly undoing the operation this command performs with a positive argument.
- With an argument of 0, this command interchanges the symbolic expressions at point and mark.

Exchange Words

Command

Keystroke: META-T

Interchanges the words before and after point, as follows:

- With a positive argument, this command interchanges the words before and after point, moves right one word, and repeats this operation the specified number of times.
- With a negative argument, this command interchanges the two words to the left of point, moves between them, and repeats this operation the specified number of times, exactly undoing the operation this command performs with a positive argument.
- With an argument of 0, this command interchanges the words at point and mark.

Simple Exchange Characters

Command

Keystroke: META-X Simple Exchange Characters

Interchanges the characters before and after point (that is, the character before the cursor and the one that the cursor is on), as follows:

- With a positive argument, this command interchanges the characters before and after point, moves right one character, and repeats this operation the specified number of times.
- With a negative argument, this command interchanges the two characters to the left of the cursor, moves between them, and repeats this operation the specified number of times, exactly undoing the operation this command performs with a positive argument.
- With an argument of 0, this command interchanges the characters at point and mark.

Reverse Following List

Command

Keystroke: META-X Reverse Following List

Reverses the elements of the list after point. Suppose you execute this command on the following list:

■ (a b c d e)

The following list shows the results:

■ (e d c b a)

Reverse Lines

Command

Keystroke: META-X Reverse Lines

Reverses the order of the specified number of lines. You specify the number of lines with a numeric argument. A positive argument reverses the order of the specified number of lines, including the line that contains the cursor. A negative argument reverses the order of the specified number of lines above the line that contains the cursor; the line that contains the cursor is not included in the operation.

Suppose you execute this command on the following lines by supplying an argument of 4. Notice that the cursor is initially on the first line.

■ a
b
c
d

The following lines show the results:

■ d
c
b
a

Fill Commands

3.19.2 The Fill commands allow you to fill text. *Fill* means to put as much text as possible on the current line without exceeding the right margin. Some of the commands also adjust text if you give them an argument. *Adjust* means to justify or add spaces until the right margins of each line align.

A *fill prefix* is a group of characters, usually blanks, that you define to start a line. You can have the prefix used automatically to start every line. Basically, the fill prefix is the left margin.

The *fill column* defines the maximum width of a line by setting the right margin. When text is filled, it cannot exceed this width.

The Fill commands allow you to perform the following operations:

- Turn on Auto Fill mode to let Zmacs automatically fill text for you.

- Fill or adjust a region.
- Fill or adjust a paragraph.
- Fill the paragraphs of multiline comments.
- Set the fill column (or maximum line width).
- Set a fill prefix to start each line.

Set Fill Column

Command

Keystroke: CTRL-X F

Sets the fill column from the current horizontal position of point. With an argument of less than 200, this command sets the fill column to the specified number of characters; otherwise, it sets the fill column to the specified number of pixels.

You can use the Where Am I command (CTRL-=) to receive information on the location of point.

Also refer to paragraph 3.19.4, Miscellaneous Text Format Commands.

Auto Fill Mode

Command

Keystroke: META-X Auto Fill Mode

Activates a minor mode in which text is filled. When the right margin is reached, text is automatically put on the next line after insertion of the fill prefix. A positive argument turns the mode on, 0 turns it off, and no argument toggles between on and off.

Also refer to the Mode commands in paragraph 3.14, Mode and Buffer Attribute Commands.

Fill Long Comment

Command

Keystroke: META-X Fill Long Comment

Fills a multiline comment's paragraphs. An entire run of comment lines is filled, each paragraph separately. The comments must begin at the start of the line. The right margin is defined by the fill column.

Fill Paragraph

Command

Keystroke: META-Q

Fills (or adjusts) this (or the next) paragraph. Point stays the same. The right margin is defined by the fill column. With a positive argument, this command adjusts rather than fills.

Fill Region Command

Keystroke: META-G

Fills (or adjusts) the region. The right margin is defined by the fill column. With a positive argument, this command adjusts rather than fills.

Set Fill Prefix Command

Keystroke: CTRL-X .

Defines the fill prefix from the current line. All of the current line up to point becomes the fill prefix. When there is a non-empty fill prefix, any line that fails to start with the fill prefix is considered a separator of paragraphs. The Fill Region command (META-G) assumes that each nonblank line starts with the prefix (which is ignored for filling purposes). To stop using a fill prefix, execute the Set Fill Prefix command at the beginning of a line.

**Lowercase and
Uppercase
Commands**

3.19.3 The Lowercase and Uppercase commands allow you to lowercase or uppercase the following items:

- Region
- Region of Lisp code
- Word

You can also uppercase the initial letters of words.

Lisp Lowercase Region Command

Keystroke: META-X Lisp Lowercase Region

Lowercases the region but does not affect strings, comments, and so on. Characters preceded by slashes in Zetalisp mode and backslashes in Common Lisp mode are also unaffected.

Lisp Uppercase Region Command

Keystroke: META-X Lisp Uppercase Region

Uppercases the region but does not affect strings, comments, and so on. Characters preceded by slashes in Zetalisp mode and backslashes in Common Lisp mode are also unaffected.

Lowercase Region Command

Keystroke: CTRL-X CTRL-L

Lowercases all text in the region.

Lowercase Word Command

Keystroke: META-L

Lowercases one or more words and moves forward over the words affected. With a negative argument, this command lowercases the words before point but does not move point.

Uppercase Initial Command

Keystroke: META-C

Lowercases one or more words but capitalizes the initial characters. This command moves forward over the words affected. With a negative argument, this command capitalizes the initial characters of the specified number of words before point but does not move point.

Uppercase Region Command

Keystroke: CTRL-X CTRL-U

Uppercases all text in the region.

Uppercase Word Command

Keystroke: META-U

Uppercases one or more words and moves forward over the words affected. With a negative argument, this command uppercases the words before point but does not move point.

**Miscellaneous Text
Format Commands**

3.19.4 The Miscellaneous Text Format commands allow you to perform the following operations:

- Center a line of text.
- Insert return characters.
- Insert form feeds (PAGE characters).
- Insert blank lines.
- Justify (adjust) a line with the right margin.
- Set a goal column for the Cursor Movement commands Up Real Line (CTRL-P) and Down Real Line (CTRL-N).
- Set the right margin (that is, the fill column).
- Split a line.

Center Line Command

Keystroke: META-S

Centers this line's text. The left margin of the line begins in the first column; the right margin is defined by the fill column. With a numeric argument, this command centers the specified number of lines and moves past them.

Insert CR Command

Keystroke: RETURN

Inserts one or more return characters into the buffer, moving point. In Auto Fill mode with no numeric argument, this command fills the line before point as well as inserting a return character. This fill operation might cause another return character to be inserted earlier in the line. If two or more blank lines follow point, this command simply moves down one line without inserting a return character.

Insert FF Command

Keystroke: META-CLEAR SCREEN

Inserts a form feed (that is, a PAGE character) into the buffer at point. This command has the same effect as CTRL-Q CLEAR SCREEN.

Make Room Command

Keystroke: CTRL-O

Inserts one or more blank lines after point. With a numeric argument, this command inserts the specified number of blank lines.

Right Adjust Line Command

Keystroke: META-X Right Adjust Line

Adjusts the current line to the right margin. With a nonzero argument, this command adjusts from point to the end of the line.

Set Fill Column Command

Keystroke: CTRL-X F

Sets the fill column from the current horizontal position of point. With an argument of less than 200, this command sets the fill column to the specified number of characters; otherwise, it sets the fill column to the specified number of pixels.

You can use the Where Am I command (CTRL-=) to receive information on the location of point.

Also refer to paragraph 3.19.2, Fill Commands.

Set Goal Column

Command

Keystroke: CTRL-X CTRL-N

Sets the goal column for the Cursor Movement commands Up Real Line (CTRL-P) and Down Real Line (CTRL-N). This command makes the current horizontal position of the cursor the goal column for the default definitions of CTRL-P and CTRL-N. CTRL-P and CTRL-N try to move to the goal column in the line to which they move. With a numeric argument, this command removes the goal column.

Split Line

Command

Keystroke: META-CTRL-O

Moves the rest of the current line down vertically. This command inserts a return character and updates the indentation of the new line to be below the old position.

Sort Commands

3.19.5 The Sort commands allow you to alphabetically sort regions by the following items:

- Lines
- Paragraphs
- Keyboard macros

Sort Lines

Command

Keystroke: META-X Sort Lines

Sorts the region alphabetically by lines.

Sort Paragraphs

Command

Keystroke: META-X Sort Paragraphs

Sorts the region alphabetically by paragraphs.

Sort via Keyboard Macros

Command

Keystroke: META-X Sort via Keyboard Macros

Sorts the region alphabetically or numerically. You are prompted in the mini-buffer for keyboard macros that are used to move to the various parts of the region to be sorted.

To make this command work properly, it is essential that the region, which you mark before entering the command, end in the proper place. The region must end exactly with the same character (returns and blanks count) that is immediately to the left of point when the end-of-record macro executes the last time. If this is not the case, no sorting is done. An error message may appear, but it is often immediately overwritten. If the sort is not done, see if the end of the region is placed correctly.

With a numeric argument, this command attempts to adjust the end of region appropriately so that the command works.

Suppose you mark the following text as a region and enter this command to sort by employee number:

Name: Winchester, Ed	Employee No.: 909456	Address: 1010 Texas
Name: Roberts, Sheila	Employee No.: 879077	Address: 413 Bluebird
Name: Ellis, Dwayne	Employee No.: 666667	Address: 1217 Rockwood
Name: Daniel, Ray	Employee No.: 884455	Address: 612 Deertree
Name: Johnson, Susan	Employee No.: 909022	Address: 1234 Parkdale
Name: Higgins, John	Employee No.: 787822	Address: 500 Main

The following procedure describes the prompts of this command:

1. You are asked in the minibuffer for the keyboard macro to move to the beginning of the sort key (in this case, the employee number). You position the keyboard cursor on the 9. Then you press CTRL-X).
2. Then you are asked for the keyboard macro to move to the end of the sort key (in this case, the end of the employee number). You position the keyboard cursor after the 6. Then you press CTRL-X).
3. Finally, you are asked for the keyboard macro to go to the end of the record, where this record ends and the next one begins. In this case, you position the keyboard cursor at the beginning of the next line. Then you press CTRL-X).

This command then sorts the region numerically by employee number.

Tab and Indentation Commands

3.19.6 Zmacs provides many Tab and Indentation commands to help you indent your Lisp programs. The indentation should reflect the structure of your Lisp program. The lines are indented according to their nesting level within parentheses.

When you execute the Indent New Line command (LINE FEED) in Common Lisp or Zetalisp mode, it moves the keyboard cursor to the next line and indents the line correctly for Lisp by adding the appropriate number of tab or space characters. Other commands allow you to perform more elaborate indentation. (The RETURN key simply takes you to the first column of the next line.)

The default tab character is equivalent to eight spaces.

The TAB key performs the indentation that is appropriate for the current mode. If you are in Text mode, the TAB key inserts a tab eight spaces long. In Common Lisp or Zetalisp mode, the TAB key inserts as many tabs as possible, and when a full tab is too much, it inserts spaces. Thus, if the TAB key needs to indent only 13 spaces, it adds one tab and five spaces.

You can change how many spaces comprise a tab by using the Set Tab Width or Tabify commands.

If you move the cursor to a tab character with the mouse, the cursor goes to the front of the tab character. The mouse box surrounds the space occupied by the tab character.

*Indentation
Commands*

3.19.6.1 The following commands allow you to control indentation.

Back to Indentation Command

Keystroke: META-M

Moves to the first nonblank character in the current line. If there is a fill prefix, this command moves to the first nonblank character after the fill prefix (even if the fill prefix is not blank).

Also refer to paragraph 3.4, Cursor Movement Commands.

Delete Indentation Command

Keystroke: META-^

Deletes the return character and any indentation at the front of the line. This command leaves a space in place of these characters where appropriate. With a numeric argument, this command moves down a line first (operating on the end of the current line and the start of the next).

In more informal terms, META-^ concatenates this line to the one above it. With a numeric argument, it concatenates this line to the one below it.

Suppose you execute this command on the following code:

```
(setf x
  (cond █ (*permanent-real-line-goal-xpos*))
```

The following line shows the results:

```
(setf x █(cond (*permanent-real-line-goal-xpos*))
```

Also refer to paragraph 3.6.1, Delete Commands.

Indent Comment Relative Command

Keystroke: META-X Indent Comment Relative

Aligns a new comment with the previous one. This command sets the Zmacs user variable Comment Column to the position of the previous comment and then executes the Indent for Comment command (CTRL-;).

Also refer to paragraph 3.12.1, Comment Commands.

Indent Differently Command

Keystroke: CTRL-TAB

Tries to indent this line differently. If called repeatedly, this command makes multiple attempts to indent differently.

Indent for Comment Command

Keystroke: CTRL-; or META-;

Moves to or creates a comment. This command finds the start of existing comments or creates a comment at the end of the current line. With a numeric argument, this command realigns existing comments for n lines but does not create any (n is the numeric argument). The following Zmacs user variables affect how this command operates; all units are raster pixels, not character counts:

- **Comment Column** — The minimum column for aligning comments. When you press the CTRL-; keystroke, the keyboard cursor goes to that column. If nothing is there, it goes to the next tab stop. You can set this column by using the Set Comment Column command (CTRL-X ;).
- **Comment Start** — The string used to recognize existing comments.
- **Comment Begin** — The string used to start new comments.
- **Comment Round Function** — The function used to compute the column for comments past the comment column.

Also refer to paragraph 3.12.1, Comment Commands.

Indent for Lisp Command

Keystroke: TAB

Indents this line as appropriate for Lisp code. A numeric argument specifies the number of lines to indent. This command applies only to Common Lisp and Zetalisp mode.

Indent Nested Command

Keystroke: META-X Indent Nested

Indents the current line for the specified nesting level. With no argument (or an argument of 1), this command indents the line at the same nesting level as the last nonblank line (that is, directly above it). A larger argument indicates that this line is the specified number of levels closer to the top level and should be indented under the last line above it whose level is the same. The previous lines are scanned under the assumption that any line indented less than its successors is one level higher than they are. However, unindented lines and comment lines are ignored. If the cursor is not at the beginning of a

line, the whole line is indented, but the cursor stays fixed with respect to the text.

Indent New Comment Line Command

Keystroke: META-LINE FEED

If the keyboard cursor is inside a comment when you press META-LINE FEED, this command continues the comment on the next line and pushes the existing next line down a line to make space for the comment. If the keyboard cursor is not inside a comment, this command acts like the Indent New Line command (LINE FEED), which inserts a return character and a tab to obtain the proper indentation on the new line.

Note that this command behaves differently depending on the number of semicolons:

- One semicolon — Indents to the comment column, which you can set with the Set Comment Column command.
- Two semicolons — Indents appropriately for Lisp code.
- Three semicolons — Does not indent.

Also refer to paragraph 3.12.1, Comment Commands.

Indent New Line Command

Keystroke: LINE FEED

Inserts a return character and properly indents the new line.

Indent New Line at Previous Sexp Command

Keystroke: META-X Indent New Line at Previous Sexp

Inserts a return character and the proper indentation at the symbolic expression before point.

Indent Region Command

Keystroke: META-CTRL-\

Indents each line in the region. The lines are indented relative to the line preceding the region. With no argument, this command calls the command currently on the TAB key to perform the indentation. (The TAB key works differently depending on the mode.) With an argument, this command indents each line the specified number of spaces in the current font.

Indent Relative Command

Keystroke: META-X Indent Relative

Indents this line relative to the previous line. With a numeric argument, this command executes the Tab to Tab Stop command; otherwise, it adds whitespace characters until the line is indented underneath an indentation point in the previous nonblank line. Successive calls to this command find successive indentation points. An indentation point is the end of a sequence of spaces and tabs. The end of the line is considered an indentation point; after the end of the line, the command cycles back to the first suitable indentation point. If there is no suitable indentation point, it executes the Tab to Tab Stop command.

Indent Rigidly Command

Keystroke: CTRL-X CTRL-I

Shifts text in the region sideways as a unit. Every line in the region has its indentation increased by the numeric argument of this command (the argument can be negative). The argument specifies a number of space characters in the default font.

Indent Sexp Command

Keystroke: META-CTRL-Q

Indents the symbolic expression that follows. Each line that starts within the symbolic expression is indented for Lisp. (The line containing point is not indented.) If the keyboard cursor is inside a symbolic expression, this command moves the text after point to the next line and aligns under that line.

Indent Under Command

Keystroke: META-X Indent Under

Properly indents a region under the previous line or under a previous string that you specify in the minibuffer. If you specify a string, the beginning of the region is indented to match the first occurrence of the string above the region. A *string* is a sequence of characters. The command searches back line by line, forward in each line, until it finds a matching string. If the string is found off the screen, it is shown in the minibuffer.

Move Over) Command

Keystroke: META-)

Moves the cursor one character past the next closing parenthesis to make room for code to be inserted. The indentation is updated, and any indentation before the closing parenthesis is deleted.

Also refer to paragraph 3.4, Cursor Movement Commands.

Stack List Vertically

Command

Keystroke: META-X Stack List Vertically

Indents the list after point, first inserting returns. For example, suppose you execute this command on the following line:

```
█ (a b c d e f)
```

The following code shows the result:

```
█(a b
  c
  d
  e
  f)
```

Now, suppose you execute this command with a numeric argument on the following line:

```
█ (a b c d e f)
```

The result is as follows:

```
(a
  b
  c
  d
  e
  f)
```

This Indentation

Command

Keystroke: META-O

Creates a new line, indented to the position of point, directly beneath the current line. With an argument of 0, this command indents the current line to the position of point. With a positive argument, this command creates a new line indented like the current line (that is, it does not consider the location of point).

Suppose you execute this command on the following line of code:

```
(a b c █ d e f)
```

The following code shows the results:

```
(a b c d e f)
```

Without an argument

```
█
```

```
█(a b c d e f)
```

With an argument of 0

```
(a b c d e f)
```

With a positive argument

```
█
```

Tab Commands 3.19.6.2 The following commands allow you to control tabs.

Edit Tab Stops Command

Keystroke: META-X Edit Tab Stops

Edits the tab-stop buffer. Once you change the tabs in the tab-stop buffer, the changes apply to all your buffers.

In the second line of the tab-stop buffer, a colon indicates a tab-stop column. When you press TAB in one of your buffers that is in Text mode, the cursor moves to this column, leaving the intervening columns blank. (For a buffer in Common Lisp or Zetalisp mode, you need to press META-X and type tab to Tab stop to move to a tab column.)

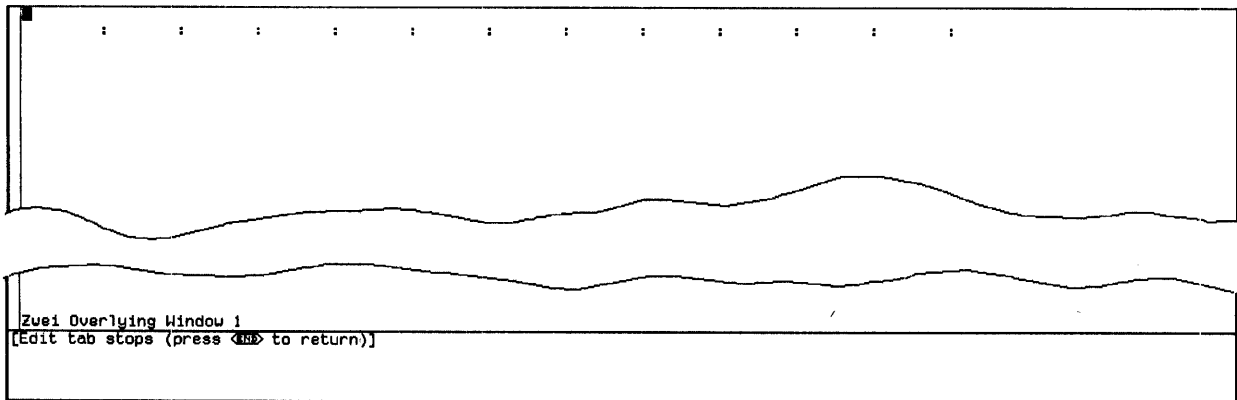
A period also indicates a tab-stop column, but in this case, the text that you place in the corresponding columns in the first line of the tab-stop buffer is copied into your buffer.

When you finish with the tab-stop buffer, press END to return to your previous buffer with the new tabs in place.

Paragraph 4.4, Login Init File, describes how to save these tab stops in your login init file and load them each time you log in.

Figure 3-27 shows the tab-stop buffer.

Figure 3-27 Tab-Stop Buffer



Insert Tab Command

Keystroke: TAB or META-TAB (depends on mode)

Inserts a tab in the buffer at point.

To determine which keystroke to use in the current mode, press HELP W.

Set Tab Width Command

Keystroke: META-X Set Tab Width

Sets the displayed width of tab characters for this buffer. This width is the separation of tab stops, measured in space characters. This change applies only to this buffer and overrides what the attribute list says. The command asks you whether to change the attribute list (-*-) in the text as well. If you supply a numeric argument with this command, it is used as the new value for the tab width. Otherwise, you are prompted in the minibuffer for the new value. The default is 8.

Also refer to paragraph 3.14, Mode and Buffer Attribute Commands.

Stupid Tab Command

Keystroke: TAB (only in Fundamental mode)

Inserts spaces from the position of point to the next even multiple of eight. The size of these spaces is determined by the current font.

Tab to Tab Stop Command

Keystroke: TAB (when in Text mode) or META-X Tab to Tab Stop

Moves the cursor to the next tab column, as specified by the tab-stop buffer, and inserts a tab character. To edit the tab-stop settings, use the Edit Tab Stops command.

Tabify Command

Keystroke: META-X Tabify

Replaces spaces with tabs in the region, or from point to the end of the buffer. All sequences of three or more spaces are replaced as much as possible with tabs, preserving the indentation. A numeric argument specifies the number of spaces that make up a tab.

Untabify Command

Keystroke: META-X Untabify

Replaces tabs with spaces in the region, or from point to the end of the buffer. All tab characters are replaced by spaces, preserving indentation. A numeric argument specifies the number of spaces that make up a tab.

Just One Space Command 3.19.7 The Just One Space command allows you to manipulate whitespace. *Whitespace* is any character contained in the Zmacs user variable `Whitespace Chars`.

Just One Space Command

Keystroke: META-X Just One Space

Replaces all whitespace around point with n spaces, where n is the numeric argument. The argument defaults to 1.

**Window
Commands**

3.20 You can split the editor buffer window into two or more windows. This division allows you to edit more than one buffer at a time or to edit the same buffer with two different windows. In addition to using the keystrokes CTRL-X O or typing META-X Other Window to move from one window to another, you can use the mouse. To select a different window with the mouse, you move the mouse cursor into that window and click left.

The following list describes some of the other operations you can perform on windows:

- Return to displaying one window.
- Scroll the other window forward or backward.
- Scroll the window until all of the current definition is in the window. This feature works on any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defun**, **deflavor**, **defvar**, and so on.
- Recenter the window.
- Make a window larger.
- Redisplay all windows. The term *redisplay* means to update the information on the screen. It removes what is there and writes it out again. Redisplaying is commonly used to remove interference.

Complete Redisplay

Command

Keystroke: CLEAR SCREEN

Redisplays all windows. By default, the text is not scrolled on the screen. However, a numeric argument specifies the screen line to which point is scrolled. A negative argument counts from the bottom of the screen.

Grow Window

Command

Keystroke: CTRL-X ^

Makes the current window (where the cursor is) larger by the number of lines specified in the numeric argument.

Modified Two Windows

Command

Keystroke: CTRL-X 4

Finds a buffer, file, jump point, or definition in the other window.

One Window

Command

Keystroke: CTRL-X 1

Returns to one-window mode. This command causes one of the editor windows in this frame to fill the whole frame. With a numeric argument, this command always uses the current window (where the blinking cursor is). Otherwise, the window is controlled by the value of the Zmacs user variable One Window Default, which can be one of the following:

- **:top** — Selects the uppermost window.
- **:bottom** — Selects the lowermost window.
- **:current** — Selects only the window in which the cursor is blinking.
- **:other** — Selects only the window in which the cursor is not blinking.

The default is **:current**.

NOTE: When you are editing compiler warnings with the Edit Warnings command (META-X Edit Warnings), Edit File Warnings command (META-X Edit File Warnings), or the Edit System Warnings command (META-X Edit System Warnings), pressing CTRL-X 1 expands the large window even if the blinking keyboard cursor is in the small window. You must supply a numeric argument with CTRL-X 1 to choose the window with the blinking cursor.

Other Window

Command

Keystroke: CTRL-X O

Moves to the other window. If there are several windows, this command cycles through them. A numeric argument specifies the window to go to, counting from 1 at the top.

You can also select another window by moving the mouse cursor to that window and clicking left. Refer to paragraph 3.15, Mouse Commands.

Recenter Window

Command

Keystroke: CTRL-L

Chooses a new point in the buffer to begin redisplay. With no argument, this command centers on the screen the line containing the point . An argument specifies the line of the window on which to put point. Negative arguments count up from the bottom of the screen.

Reposition Window

Command

Keystroke: META-CTRL-R

Attempts to display all of the current definition in the window. The definition can be any defining construct that begins with an opening parenthesis in the first column followed by **def**, such as **defun**, **defflower**, **defvar**, and so on. The command works as follows:

- If the beginning of the definition is on the screen, this command scrolls forward to bring the end onto the screen.
- If the beginning of the definition is off the screen, this command scrolls backward so that the beginning appears, but it does not push point off the bottom of the screen.
- If the beginning of the definition is at the top of the screen, this command tries omitting or including the comments before it.
- If the definition is entirely on the screen, this command positions it at the top of the screen. (With a numeric argument, the definition is positioned at the bottom of the screen.)

Also refer to paragraph 3.17, Scroll Commands.

Scroll Other Window

Command

Keystroke: META-CTRL-V

Scrolls the other window forward several lines. A numeric argument specifies the number of lines. A negative argument scrolls backward. The default scrolls one screen forward.

Also refer to paragraph 3.17, Scroll Commands.

Split Screen

Command

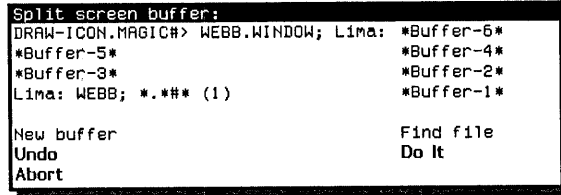
Keystroke: META-X Split Screen

Makes several windows split among the buffers as specified by menu selections. This command provides the option to create new buffers and edit new files.

Figure 3-28 shows the display that appears when you execute the Split Screen command.

Figure 3-28

Split Screen Display (First Screen)



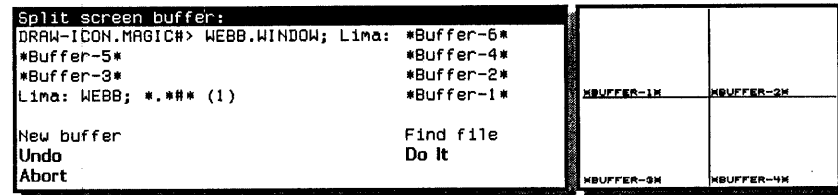
You can choose one of the buffers listed by clicking on its name, select *New buffer* to create a new buffer, or select *Find file* to bring a file into a buffer for editing. When you specify the buffer or file, an empty box appears beside the Split Screen display with the buffer name on the bottom. When you specify the next buffer you want, the box is split in half. This box is divided among the buffers until you click on one of the following:

- **Do It** — Executes the command.
- **Undo** — Allows you to undo the buffers displayed. It removes one buffer name at a time from the box, starting with the last buffer you chose.
- **Abort** — Aborts the Split Screen command.

Suppose you want to split the screen among four buffers. Figure 3-29 shows an example Split Screen display after you have chosen four buffers.

Figure 3-29

Split Screen Display (Second Screen)



When you finish executing the Split Screen command, you can perform the following operations:

- Return to displaying only one window by using the *One Window* command (CTRL-X 1). The window displayed is the one that the blinking keyboard cursor is in.
- Move to another window. You can use two methods to move to another window:
 - Execute the *Other Window* command (CTRL-X O). If several windows are on the screen, this command goes through them all in cyclic order. A numeric argument specifies the window to go to, counting from 1 at the top.
 - Move the mouse cursor to another window and click any button.

- Scroll another window by using the Scroll Other Window command (META-CTRL-V). This command scrolls the other window forward one screen of text. If several windows are on the screen, this command finds the other window to scroll by going from left to right or by going back to the top left window if you are in the lower right window. A numeric argument specifies to scroll forward by that number of lines. A negative argument scrolls backward.

Two Windows

Command

Keystroke: CTRL-X 2

Selects two windows. This command splits the frame into two editor windows and selects the bottom one. With a numeric argument, this command makes the second window point to the same buffer that the first window does.

Two Windows Showing Region

Command

Keystroke: CTRL-X 8

Makes two windows on the same buffer. The top window shows the current region.

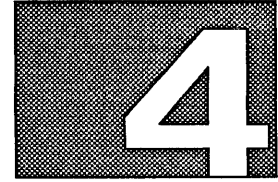
View Two Windows

Command

Keystroke: CTRL-X 3

Selects two windows but stays in the first one. This command splits the frame into two editor windows and selects the top one. With a numeric argument, this command makes the second window point to the same buffer that the first window does.

CUSTOMIZING ZMACS



Introduction

4.1 This section describes how you can customize Zmacs to suit your particular needs. The following topics are discussed:

- Changing user variables — Describes how to find and change Zmacs user variables.
- Customizing keys — Discusses installing a command on a key.
- Login init files — Describes how to make a login init file to set up Zmacs the way you want it when you log in.
- Writing your own commands — Describes how to write your own commands in Zmacs.
- Standalone editor — Tells how to use a standalone editor to edit information in your program.

You can also customize Zmacs with major modes, minor modes, and buffer attributes. These topics are discussed in paragraph 2.12, *Setting Modes and Buffer Attributes*.

The Edit Screen option on the System menu allows you to change the shape of the editor window. The *Introduction to the Explorer System* describes how to use the Edit Screen option.

Changing User Variables

4.2 Zmacs provides user variables to allow you to customize Zmacs. You can easily find and change these variables to the values you want.

Paragraph 3.5.3, *Variable Commands*, describes each of the commands that work on user variables. These commands allow you to perform the following operations:

- List Variables (META-X List Variables) — Lists all user variables and their values.
- Variable Apropos (HELP V) — Helps you find and change a user variable.
- Describe Variable (HELP O) — Provides documentation about a user variable.
- Kill Local Variable (META-X Kill Local Variable) — Changes a user variable that is local to the buffer into a global variable.
- List Local Variables (META-X List Local Variables) — Lists user variables local to the buffer.

- **Make Local Variable (META-X Make Local Variable)** — Makes a user variable local to the buffer.
- **Set Variable (META-X Set Variable)** — Sets a user variable.

Subsequent paragraphs describe how to list and change user variables.

Listing and Changing User Variables

4.2.1 The **List Variables** command (META-X List Variables) lists all the user variables and their values. With a numeric argument, it provides documentation as well. However, you cannot change the variables via this display.

The **Variable Apropos** command (**HELP V**) provides the following options:

- If you press **HELP V** and type a string, it lists all user variables whose names contain a given string. It provides a line of documentation on each variable.
- If you press **HELP V** and then press **RETURN**, it also lists all the user variables.
- The extended search characters are available with **HELP V**. They allow you to perform complicated searches for combinations of strings. You can see a list of them when you are executing this command by pressing **CTRL-H HELP**.
- The display given by **HELP V** is mouse-sensitive to allow easy modification of variable values. When you click left on any variable listed, its value appears for editing in the minibuffer. When you have the new value in the minibuffer, you press the **END** key and the variable is changed.

You can also click right on any variable listed to invoke a menu that provides more documentation on the variable or that allows you to modify the variable.

NOTE: When **HELP V** lists a user variable, such as **Center Fraction**, the actual variable name is **zwei:*center-fraction***.

Examples of Changing User Variables

4.2.2 The following list describes some user variables that you may want to change:

Check Unbalanced Parentheses When Saving

When this user variable is set to **t**, Zmacs checks for unbalanced parentheses when saving a Common Lisp mode or Zetalisp mode buffer. The default is **nil**.

Region Marking Mode

This user variable specifies how Zmacs should mark a region. Two options are available: **:underline** and **:reverse-video**. When the variable is set to **:underline**, Zmacs underlines a region. When it is set to **:reverse-video**, Zmacs displays the region in reverse video. The default is **:underline**.

Sticky Minor Modes

This user variable specifies which minor modes to carry from the current buffer to new buffers. The default contains **atom-word-mode** and **word-abbrev-mode**.

Initial Minor Modes

This user variable specifies which minor modes should automatically be activated in any major mode. The default is **nil**.

Customizing Keys

4.3 Command tables define command-to-key (and mouse) associations. When you associate a key sequence (or mouse click) with a command using the Install Command command (META-X Install Command), the Install Macro command (META-X Install Macro), the Install Mouse Macro command (META-X Install Mouse Macro), or the Set Key command (META-X Set Key), you should be aware of how command tables work. The following items describe the use of command tables:

- Command tables differ depending on the context in which you use them. For example, if you want the command to be on a key prefixed by CTRL-X, you need to put it in **zwei:*zmacs-control-x-comtab***. The variable **zwei:every-comtab** provides a list of all the command tables.
- If another command is already associated with the key sequence (or mouse click) you want to use, you are warned of this situation.

When you associate a key (or mouse click) with a command, that association lasts only until you cold boot, unless you add a system patch or put the association in your login init file. When you cold boot, you return to the associations that were originally in the operating system.

Paragraph 3.5, Customization Commands, discusses the Install Command command, the Install Macro command, the Install Mouse Macro command, and the Set Key command.

Login Init File

4.4 The login initialization (init) file can initialize Zmacs by executing whatever Lisp functions you specify. The system executes the init file whenever you log in. You can use the init file to put different commands on different keys. *Explorer Tools and Utilities* explains login init files in detail.

The following procedure describes how to create a login init file:

1. Create a buffer.
2. Put your initialization functions in it.
3. Save the buffer to a file named LOGIN-INIT in your directory (that is, the directory with the same name as your login user ID).

If your user ID is JONES, then your login init file is named JONES; LOGIN-INIT.

4. If you want, compile the file in order to store an xld version. The system uses the xld version instead of the Lisp version regardless of the dates.

All changes in the login init file should *always* be inside a `login-forms` form. The following code shows an example:

```
(login-forms
  (setf zwei:*region-marking-mode* :reverse-video)
  (setf zwei:*check-unbalanced-parentheses-when-saving* t)
  (zwei:set-comtab zwei:*standard-comtab*
    `(#\s-m-f zwei:com-function-apropos))
)
```

This example changes the display of a marked region from underlining to reverse video. It checks for unbalanced parentheses when saving a Common Lisp mode or Zetalisp mode buffer.

set-comtab *comtab specs* &optional *extended-commands* Function

This function allows you to store a command on a key sequence.

Arguments: *comtab* — This argument specifies the command table where you want the key stored: `zwei:*standard-comtab*`.

specs — This argument specifies a list with an even number of elements that are alternating keys and commands. The command needs to be a function name, such as `zwei:com-function-apropos`.

You represent keys in Lisp code with `#\` followed by the key names. You can abbreviate the following key names if these abbreviations are followed by a hyphen. That is, `#\c` is simply a lowercase `c`, whereas `#\c-a` is CTRL-A.

Key	Abbreviation
CTRL	c
META	m
SUPER	s
HYPER	h

extended-commands — This optional argument specifies an extended command association list, which is a list of dotted pairs of strings and commands.

Example:

```
zwei:(set-comtab *standard-comtab*
  `(#\s-k com-kill-or-save-buffers
    #\s-t com-select-system-as-tag-table)
  `(("set fill column" . com-set-fill-column)))
```

After evaluating this example, you can press SUPER-K to invoke the Kill or Save Buffers command and SUPER-T to invoke the Select System as Tag Table command. Also, you can invoke the Set Fill Column command by typing META-X Set Fill Column.

You can also save the tab stops that you create with the Edit Tab Stops command in your login init file. To do this, you use the following function:

zwei:load-tab-stop-buffer *name-string1 name-string2* Function

This function allows you to save the tab stops you create in the tab-stop buffer and load them in your login init file.

- Arguments:* *name-string1* — This argument contains the text, enclosed in double quotes, that appears in the first line of your tab-stop buffer.
- name-string2* — This argument contains the colons and periods, enclosed in double quotes, that appear in the second line of your tab-stop buffer.

Writing Your Own Commands

4.5 The system makes no distinction between system commands and user commands. Any command you write in the editor and compile is just as valid and effective as any system-defined command (that is, until you cold boot). You can redefine and change any command in the system.

If you want the change to be permanent, you need to store it in a file and load it in your login init file (the change is for you only). You can also create a patch to save your commands. Refer to paragraph 3.11.6, Patch Commands.

The following procedure describes how to put your own commands into the system:

1. Write your command.

When you write your command, you should probably use `zwei:defcom`. You can use the Edit Definition command (META-.) to find `zwei:defcom` and also to find examples of other commands in the editor to see how `zwei:defcom` is used.

When you are defining your command in the editor, you may want to put it in the ZWEI package (the editor package). Then your command does not need `zwei:` in front of all variables and functions it references in the ZWEI package. You specify the ZWEI package by executing the Set Package command (META-X Set Package). You can make your own package, but it is easier to put the command in ZWEI.

2. Use the function (`fboundp 'function-name`) to make sure no one else is already using a command by that name.
3. Store the command in a file and load it in your login init file, or patch the command.

You can also modify a Zmacs command. The Edit Definition command (META-.) finds the definition of a specified command and allows you to edit it. Use the Edit Zmacs Command command (META-CTRL-.) to edit the command installed on a specified key.

If there are multiple definitions of a command, a list of all definitions is kept in the `*DEFINITIONS*` buffer. When you are in this buffer and the cursor is on the appropriate line, the CTRL-/ command takes you to the definitions.

Editor functions that are bound to keys are usually defined with the form `zwei:defcom` rather than `defun`. The `defun` form works if you define the function with an empty parameter list. Regardless of which defining function you use, the function you write must return a numeric code to inform Zmacs of the amount of redisplay necessary on the screen after your function runs. Possible values include `zwei:dis-none`, `zwei:dis-bps`, and `zwei:dis-all`. Refer to the online documentation for the function `zwei:must-redisplay` for a complete list and explanation of appropriate values.

Standalone Editor

4.6 You can use a standalone editor to edit information in your programs. A *standalone editor* is an editor that provides almost all of the features of a Zmacs editor without being tied to the rest of the editing system. It does not access the list of buffers that normal editors do; however, you can use mini-buffers in it.

You can make a standalone editor window by setting a variable such as **editor*** to the result of the function **make-instance** of the **zwei:standalone-editor-window** flavor. The following code shows how you write this for your program:

```
(setf editor* (make-instance `zwei:standalone-editor-window))
(setf users-input (edstring "initial editor contents" editor*))
```

When the code executes, **editor*** becomes an editor. If you use the function **edstring**, you can give **edstring** a string that you want displayed as the initial contents in the editor. You might want nothing displayed. The person using the program types in what is wanted, and **edstring** returns the value that is entered.

For information on how to manipulate standalone editor windows, refer to the *Explorer Window System Reference*.

All standalone editors use **zwei:*standalone-comtab***.

INDEX

Introduction

The indexes for this Explorer software manual are divided into two sub-indexes: a general subindex and a command subindex. The pages on which they begin are as follows:

Index Name	Page
General	Index-1
Command	Index-7

General**A**

Abort commands, 3-105
 overview, 2-14
 accessing Zmacs, 2-2
 adjusting text, 3-182
 Any Bracket mode, 3-145
 Append to Buffer command, 3-8
 Append to File command, 3-97
 apropos
 command, 3-106
 function, 3-109
 user variable, 3-112
 Atom Word mode, 3-145
 attribute bits, 2-43
 attribute list
 definition, 2-32
 reparsing, 3-147
 updating, 3-147
 Auto Fill mode, 3-145, 3-182

B

backspace buffer attribute, 3-148
 base buffer attribute, 3-148
 block, 3-77
 Break command, 3-117
 buffer attributes
 backspace, 3-148
 base, 3-148
 commands for setting, 3-147—3-151
 fonts, 3-149
 lowercase, 3-149
 nofill, 3-149
 package, 3-149
 patch file, 3-150
 tab width, 3-150
 VSP (vertical interline spacing), 3-150

Buffer commands, 3-7—3-20
 Append to Buffer command, 3-8
 Capture Into Buffer commands, 3-7
 Edit and List Buffer commands, 3-10—3-15
 Edit and List Changed Definition
 commands, 3-15—3-17
 Insert Buffer commands, 3-8
 Kill and Save Buffer commands, 3-8—3-10
 List and Edit Buffer commands, 3-10—3-15
 List and Edit Changed Definition
 commands, 3-15—3-17
 List Modifications of Buffer command, 3-18
 Print Buffer commands, 3-19
 Read-Only command, 3-18
 Rename Buffer command, 3-18
 Revert Buffer command, 3-20
 Save and Kill Buffer commands, 3-8—3-10
 Sectionize Buffer command, 3-18
 Select Buffer commands, 3-20
 View Buffer command, 3-19
 buffers
 definition, 2-15
 header, 2-19
 history, 3-13
 how buffers handle Lisp code, 2-19
 overview on creating, 2-20

C

Caller commands, 3-117—3-119
 Capture Into Buffer commands, 3-7
 centering a line, 3-186
 Change File Properties commands, 3-93
 character, 2-24
 command. *See* Zmacs commands
 command tables, Zmacs, 4-3
 Comment commands, 3-133—3-135
 Common Lisp mode, 3-143
 Compare Source commands, 3-128

Compile and Evaluate commands, 3-21—3-32
 Compile and Load commands, 3-28
 Compile File command, 3-27
 Compile or Evaluate Buffer commands, 3-26
 Compile or Evaluate Changed Definition commands, 3-25
 Compile or Evaluate Region or Definition commands, 3-23
 Compiler Warning commands, 3-29—3-31
 Disassemble commands, 3-32
 Evaluate Into Buffer commands, 3-23
 Evaluate Minibuffer command, 3-23
 Update Xld command, 3-28
 Completion commands, 3-3—3-5
 Copy File commands, 3-94
 Count commands, 3-141
 CTRL-H, 3-163
 CTRL-Q, 3-45
 CTRL-X, 3-3
 cursor location (point), 2-27
 Cursor Movement commands, 3-33—3-46
 General Cursor Movement commands, 3-33—3-35
 general rules for cursor movement, 2-24
 Lisp Cursor Movement commands, 3-35—3-39
 listing of commands grouped by quantity, 2-26
 Saving Cursor Locations (Point) commands, 3-41—3-44
 Text Cursor Movement commands, 3-40
 Various Quantities command, 3-45
 Customization commands, 3-47—3-58
 Install Command on Key commands, 3-47
 Keyboard Macro commands, 3-48—3-51
 Variable commands, 3-52—3-54
 Word Abbreviation commands, 3-54—3-58
 customizing Zmacs, 4-1—4-6

D

Date command, 3-142
 zwei:defcom function, 4-5
 definition, 2-24
 Delete File commands, 3-95
 Deleting and Moving Text commands, 3-59—3-74
 Delete commands, 3-59—3-61
 Kill commands, 3-61—3-65
 Mark commands, 3-65—3-67
 overview, 2-34—2-44
 Register commands, 3-68—3-72
 Yank (Retrieve) commands, 3-72—3-74
 deleting text
 commands, 3-59—3-61
 definition of, 2-34
 directory
 definition, 2-15
 subdirectory, 3-87

Directory commands, 3-75—3-92
 Create, List, and Clean Directory commands, 3-75—3-76
 Edit Directory (Dired) commands, 3-77—3-92
 Dired commands, 3-77—3-92
 dired function, 2-3
 disassembling compiled code, 3-32

E

ed function, 2-2
 Edit and List Buffer commands, 3-10—3-15
 Edit and List Changed Definition commands, 3-15—3-17
 Edit Source commands, 3-119
 editor buffer window, 2-3
 Editor menu, Top-Level, 2-11
 zwei:edstring function, 4-6
 Electric Font Lock mode, 3-145
 Electric Shift Lock mode, 3-145
 entering Zmacs, 2-2
 epsilon character, 3-101
 Evaluate commands. *See* Compile and Evaluate commands
 Exchange (Transpose) commands, 3-179—3-182
 exiting Zmacs, 2-3
 extended search, 3-163

F

File commands, 3-93—3-100
 Append to File command, 3-97
 Change File Properties command, 3-93
 Copy File commands, 3-94
 Delete File commands, 3-95
 Find and View File commands, 3-96
 Insert File commands, 3-97
 Print File command, 3-98
 Rename File commands, 3-98
 Save and Write File commands, 3-99
 files
 appending, 3-97
 changing properties of, 3-93
 compiling, 3-27
 definition, 2-15
 hogs, 3-84
 loading, 3-28
 login initialization file for Zmacs, 4-3
 mode line information, 2-18
 pathname structure, 2-16
 properties, 3-89
 types, 2-18
 version limit, 3-95
 viewing, 3-96
 visiting, 3-96
 wildcard, 3-93
 fill column, 3-182

Fill commands, 3-182—3-184
 fill prefix, 3-182
 filling text, 3-182—3-184
 Find and View File commands, 3-96
 Flavor commands, 3-120—3-122
 Font commands, 3-101—3-104
 function
 a-propos, 3-109
 compiling a, 3-21—3-32
 evaluating a, 3-21—3-32
 help on finding a, 3-109—3-112
 tracing, 3-132
 Zmacs. *See* Zmacs functions
 Fundamental mode, 3-144

G

global user variable, 3-52
 global word abbreviations, 3-54
 goal column, 3-187
 Grind (Pretty Print) commands, 3-136

H

Help, Documentation, and Undo commands,
 3-105—3-116
 Abort commands, 3-105
 Command and Key Help commands,
 3-106—3-108
 Function and Variable Help commands,
 3-109—3-112
 Help Menu commands, 3-113
 List Keystroke History command, 3-114
 Minibuffer Help commands, 3-114
 overview of commands for Lisp code, 2-13
 Symbol Help command, 3-115
 Teach Zmacs command, 3-115
 troubleshooting, 2-14
 Undo commands, 3-115
 Where Am I command, 3-115
 hogs, 3-84

I

incremental search, 3-161
 secondary, 3-161
 Indentation and Tab commands, 3-188—3-195
 Insert Buffer commands, 3-8
 Insert File commands, 3-97
 Insert mode. *See* Overwrite mode
 Install Command on Key commands, 3-47
 interning a symbol, 3-122
 interpreting a function, 3-21

J

justifying text, 3-182

K

keyboard macro, 3-48

interactive, 3-51
 permanent, 3-50—3-51
 sorting, 3-187
 Keyboard Macro commands, 3-48—3-51
 keystroke
 help on finding a, 3-106—3-108
 history, 3-114
 installing commands on a, 3-47
 Kill and Save Buffer commands, 3-8—3-10
 kill history
 commands, 3-59—3-74
 definition of, 2-34
 killing text
 commands, 3-61—3-65
 definition of, 2-34

L

line
 centering, 3-186
 creating a new, 3-186
 sorting, 3-187
 Lisp code
 compiling, 3-21—3-32
 cursor movement, 3-35—3-39
 evaluating, 3-21—3-32
 how buffers handle Lisp code, 2-19
 Lisp Programming commands, 3-117—3-132
 Break, 3-117
 Caller commands, 3-117—3-119
 Edit or Find Source commands, 3-119
 Flavor commands, 3-120—3-122
 Package command, 3-122
 Patch commands, 3-124—3-126
 Possibility commands, 3-126—3-128
 Source Compare commands, 3-128—3-131
 Trace command, 3-132
 Lisp Syntax commands, 3-133—3-140
 Comment commands, 3-133—3-135
 Grind (Pretty Print) commands, 3-136
 Macro Expansion commands, 3-136
 Parentheses commands, 3-137—3-140
 List and Edit Buffer commands, 3-10—3-15
 List and Edit Changed Definition commands,
 3-15—3-17
 zwei:load-tab-stop-buffer function, 4-4
 local user variable, 3-52
 login initialization file for Zmacs, 4-3
 login-forms function, 4-4
 Lowercase and Uppercase commands, 3-184
 lowercase buffer attribute, 3-149

M

macro
 expansion, 3-136
 tracing, 3-132
 Macro Expansion commands, 3-136
 Macsyma mode, 3-144

major mode
 commands, 3-143
 Common Lisp mode, 3-143
 definition of, 2-30
 Fundamental mode, 3-144
 Macsyma mode, 3-144
 Text mode, 3-143
 Zetalisp mode, 3-143
 Ztop mode, 3-144

margin, right (setting the fill column), 3-183

marking text
 commands, 3-65—3-67
 overview on, 2-36
 unmarking, 3-65

menus, 2-9—2-11

META-CTRL-X, 3-2

META-X, 3-2

minibuffer, 2-5
 commands, 3-3—3-5
 evaluating Lisp code in the, 3-23
 messages in, 3-3

minor mode
 Any Bracket mode, 3-145
 Atom Word mode, 3-145
 Auto Fill mode, 3-145
 commands, 3-144—3-147
 definition of, 2-31
 Electric Font Lock mode, 3-145
 Electric Shift Lock mode, 3-145
 Overwrite mode, 3-146
 RETURN Indents mode, 3-146
 sticky, 2-31
 unsticky, 2-31
 Uppercase Global Functions mode, 3-146
 Word Abbreviation mode, 3-146

Miscellaneous commands, 3-141

Mode and Buffer Attribute commands,
 3-143—3-150
 Buffer Attribute commands, 3-147—3-151
 Major Mode commands, 3-143
 Minor Mode commands, 3-144—3-147

mode line, 2-4

mode line window, 2-4

mode word abbreviations, 3-54

Mouse commands, 3-151

mouse documentation window, 2-5

N

nofill buffer attribute, 3-149

numeric arguments, 3-5

O

Overwrite mode, 3-146

P

Package command, 3-122

PAGE character, 3-45

paragraph, 2-24
 sorting, 3-187

Parentheses commands, 3-137—3-140
 finding unbalanced parentheses, 3-138

Patch commands, 3-124—3-126

patch file buffer attribute, 3-150

pathname structure, 2-16

point (cursor location), 2-27
 command to show the location of, 3-115

point PDL, 2-27

POSSIBILITIES LISTS buffer, 3-110

Possibility commands, 3-126—3-128

Prefix command, 3-3

Pretty Print (Grind) commands, 3-136

Print Buffer commands, 3-19

Print commands, 3-153

Print File command, 3-98

program development, 1-2

Q

quantity, 3-45

R

real line, 2-24

region, definition of, 2-36

register
 overview on storing a cursor location in a,
 2-29
 overview on storing text in a, 2-43

Register commands, 3-68—3-72

Rename Buffer command, 3-18

Rename File commands, 3-98

reparing the attribute list, 3-147

replace
 multiple query, 3-170
 query, 3-170
 string, 3-171
 tag tables, 3-172—3-179

Replace commands, 3-168—3-171
 Tag Table commands, 3-172—3-179

return character, 3-186

RETURN Indents mode, 3-146

reverse assembling code, 3-32

Revert Buffer command, 3-20

right margin (setting the fill column), 3-183

S

Save and Kill Buffer commands, 3-8—3-10

Save and Write File commands, 3-99

scroll bar, 2-5

Scroll commands, 3-155—3-158

search
 extended, 3-163
 incremental, 3-161
 string, 3-159
 tag tables, 3-172—3-179

Search commands, 3-159—3-168
 Tag Table commands, 3-172—3-179
 secondary incremental search, 3-161
 section, 2-19
 Sectionize Buffer command, 3-18
 Select Buffer commands, 3-20
 semicolon, 3-133
 sentence, 2-24
 set-comtab function, 4-4
 Source Compare commands, 3-128
 splitting the screen, 3-197—3-201
 standalone editor, 4-6
 string search, 3-159
 subdirectory, 3-87
 Suggestions menus for Zmacs, 2-11
 symbol
 help, 3-115
 interning, 3-122
 symbolic expression, 2-24
 System menu, 2-10
 system status line, 2-5

T

Tab and Indentation commands, 3-188—3-195
 TAB key, 3-188
 tab width buffer attribute, 3-150
 tab-stop buffer, 3-194
 saving tab stops in a login initialization file,
 4-4
 Tag Table commands, 3-172—3-179
 text
 adjusting text, 3-182
 cursor movement, 3-40
 deleting text, 3-59—3-61
 definition of, 2-34
 editing text overview, 1-2
 filling text, 3-182—3-184
 justifying text, 3-182
 kill history, 3-59—3-74
 definition of, 2-34
 killing text, 3-61—3-65
 definition of, 2-34
 marking text, 3-65—3-67
 overview on, 2-36
 region, definition of, 2-36
 storing text in registers, 3-68
 text storage facilities overview, 1-4
 unmarking text, 3-65
 yanking (retrieving) text, 3-72—3-74
 Text Format commands, 3-179—3-196
 Exchange commands, 3-179—3-182
 Fill commands, 3-182—3-184
 Just One Space command, 3-196
 Lowercase and Uppercase commands, 3-184

Miscellaneous Text Format commands,
 3-185—3-187
 Sort commands, 3-187
 Tab and Indentation commands,
 3-188—3-195
 Text mode, 3-143
 Top-Level Editor menu, 2-11
 Trace command, 3-132
 Transpose (Exchange) commands,
 3-179—3-182
 troubleshooting Zmacs, 2-14
 tutorial, Zmacs, 3-115
 timeout window, 2-6
 removing, 2-6

U

Undo commands, 3-115
 updating the attribute list, 3-147
 Uppercase and Lowercase commands, 3-184
 Uppercase Global Functions mode, 3-146
 user variable
 apropos, 3-112
 commands, 3-52—3-54
 examples of changing a, 4-1—4-3
 global, 3-52
 help on finding a, 3-109—3-112
 local, 3-52

V

Variable commands, 3-52—3-54
 version limit for files, 3-95
 vertical interline spacing (VSP), 3-150
 View and Find File commands, 3-96
 View Buffer command, 3-19
 viewing files, 3-96
 visiting files, 3-96
 VSP (vertical interline spacing) buffer attribute,
 3-150

W

whitespace, 3-196
 wildcard for pathnames, 3-93
 Window commands, 3-197—3-201
 word, 3-54
 Word Abbreviation commands, 3-54—3-58
 Word Abbreviation mode, 3-146
 Write and Save File commands, 3-99

X

xld file updating, 3-28

Y

yanking text, commands, 3-72—3-74

Z

Zetalisp mode, 3-143

Zmacs

accessing, 2-2

command tables, 4-3

customization, 4-1—4-6

entering, 2-2

exiting, 2-3

menus, 2-9—2-11

overview, 1-1—1-8

screen, 2-3—2-6

standalone editor, 4-6

Suggestions menus, 2-11

Top-Level Editor menu, 2-11

troubleshooting, 2-14

tutorial, 3-115

Zmacs commands

apropos, 3-106

command groups, 3-1—3-201

extended commands, 3-2

help on finding, 3-106—3-108

how to execute, 3-2—3-6

numeric arguments, 3-5

writing your own, 4-5

Zmacs functions

zwei:defcom, 4-5

direc, 2-3

ed, 2-2

zwei:edstring, 4-6

zwei:load-tab-stop-buffer, 4-4

login-forms, 4-4

set-comtab, 4-4

Zmacs screen

editor buffer window, 2-3

minibuffer, 2-5

mode line, 2-4

mode line window, 2-4

mouse documentation window, 2-5

scroll bar, 2-5

splitting the screen, 3-197—3-201

system status line, 2-5

timeout window, 2-6

removing, 2-6

Ztop mode, 3-144

Commands
A

Abort at Top Level (ABORT), 3-106
 Abort One Level (CTRL-G), 3-65, 3-106
 Add Global Word Abbrev (CTRL-X +), 3-55
 Add Mode Word Abbrev (CTRL-X CTRL-A),
 3-55
 Add Patch (META-X), 3-125
 Add Patch Buffer Changed Definitions
 (META-X), 3-125
 Add Patch Changed Definitions (META-X),
 3-125
 Any Bracket Mode (META-X), 3-145
 Append Next Kill (META-CTRL-W), 3-62
 Append to Buffer (CTRL-X A), 3-8
 Append to File (META-X), 3-97
 Apropos (HELP A), 3-106
 Arglist (META-X), 3-109
 Atom Query Replace (META-X), 3-169
 Atom Word Mode (META-X), 3-145
 Auto Fill Mode (META-X), 3-145, 3-183

B

Back to Indentation (META-M), 3-36, 3-189
 Backward Character (CTRL-B or ←), 3-33
 Backward Down List (META-X), 3-36
 Backward Kill Sentence (CTRL-X RUBOUT),
 3-62
 Backward Kill Sexp (META-CTRL-RUBOUT),
 3-62
 Backward Kill Sexp No Up (META-X), 3-62
 Backward Kill Word (META-RUBOUT), 3-63
 Backward List (META-CTRL-P), 3-36
 Backward Paragraph (META-[]), 3-40
 Backward Sentence (META-A or META-↑),
 3-40
 Backward Sexp (META-CTRL-B or
 META-CTRL-←), 3-37
 Backward Sexp No Up (META-X), 3-37
 Backward Up List (META-CTRL-(), 3-37
 Backward Word (META-B or META-←), 3-40
 Beginning of Definition (META-CTRL-[or
 META-CTRL-↑), 3-37
 Beginning of Line (CTRL-A or SUPER-←),
 3-33
 Break (BREAK), 3-117
 Buffer Edit (META-X), 3-11

C

Call Last Kbd Macro (CTRL-X E), 3-50
 Cancel Patch (META-X), 3-125
 Center Line (META-S), 3-186
 Change Default Font (META-CTRL-J), 3-102
 Change File Properties (META-X), 3-93

Change Font Char (CTRL-J), 3-102
 Change Font Region (CTRL-X CTRL-J), 3-103
 Change Font Word (META-J), 3-103
 Change One Font Region (META-SHIFT-J),
 3-104
 Clean Directory (META-X), 3-75
 Close Definition (META-X), 3-137
 Comment Out Region (CTRL-X CTRL-;),
 3-133
 Common Lisp Mode (META-X), 3-143
 Compile and Exit (META-Z), 3-27
 Compile and Load File (META-X), 3-28
 Compile Buffer (META-X), 3-27
 Compile Buffer Changed Definitions
 (META-X), 3-26
 Compile Changed Definitions (META-X), 3-26
 Compile File (META-X), 3-27
 Compile Region (CTRL-SHIFT-C), 3-25
 Complete Redisplay (CLEAR SCREEN), 3-197
 Compose Character (META-CTRL-C), 3-104
 Continue Tags Multiple Query Replace
 (META-X), 3-174
 Continue Tags Query Replace (META-X),
 3-175
 Copy Binary File (META-X), 3-94
 Copy File (META-X), 3-94
 Copy From Previous Line (META-X), 3-72
 Copy Text File (META-X), 3-94
 Count Lines (META-X), 3-141
 Count Lines Page (CTRL-X L), 3-141
 Count Lines Region (META-=), 3-141
 Count Occurrences (META-X), 3-141
 Count Words (META-X), 3-141
 Create Directory (META-X), 3-75
 Current Tag Table (META-X), 3-172

D

Define Word Abbrevs (META-X), 3-55
 Deinstall Macro (META-X), 3-51
 Delete () (META-X), 3-61, 3-137
 Delete Backward (RUBOUT), 3-59
 Delete Blank Lines (CTRL-X CTRL-O), 3-60
 Delete File (META-X), 3-95
 Delete Forward (CTRL-D), 3-60
 Delete Horizontal Space (META-\), 3-60
 Delete Indentation (META-^), 3-60, 3-189
 Delete Matching Lines (META-X), 3-60,
 3-169
 Delete Non Matching Lines (META-X), 3-61,
 3-169
 Describe Command (HELP D), 3-107
 Describe Flavor (META-X), 3-121
 Describe Variable (HELP O), 3-53, 3-109

Describe Variable at Point (CTRL-SHIFT-V), 3-109
 Dired (CTRL-X CTRL-D), 3-80
 Dired Abort (ABORT), 3-80
 Dired Apply Function (A), 3-83
 Dired Automatic (H), 3-85
 Dired Automatic All (H with a numeric argument), 3-85
 Dired Change File Properties (.), 3-90
 Dired Compile File (B), 3-87
 Dired Complement Don't Delete (@), 3-88
 Dired Complement Don't Supersede (#), 3-88
 Dired Complement No Reap Flag (\$), 3-88
 Dired Copy (C), 3-86
 Dired Delete (D), 3-83
 Dired Documentation (HELP), 3-80
 Dired Edit File (E), 3-81
 Dired Edit File Two Windows (CTRL-SHIFT-E), 3-81
 Dired Edit Superior Directory (<), 3-81
 Dired Execute (X), 3-86
 Dired Exit (Q), 3-86
 Dired Find File (F), 3-82
 Dired Load File (L), 3-87
 Dired Next Hog (N), 3-85
 Dired Next Undumped (I), 3-88
 Dired Print File (P), 3-83
 Dired Print File Attributes (,), 3-89
 Dired Rename (R), 3-86
 Dired Reverse Undelete (RUBOUT), 3-84
 Dired SRCCOM (=), 3-90
 Dired Subdirectory (S), 3-87
 Dired Undelete (U), 3-84
 Dired View File (V), 3-82
 Disassemble (META-X), 3-32
 Display Directory (CTRL-X D), 3-76
 Display Font (META-X), 3-104
 Documentation (HELP), 3-113
 Down Comment Line (META-N), 3-37, 3-134
 Down Indented Line (META-X), 3-38
 Down List (META-CTRL-D), 3-38
 Down Real Line (CTRL-N or ↓), 3-34

E

Edit Buffer Changed Definitions (META-X), 3-15
 Edit Buffers (META-X), 3-11
 Edit Callers (META-X), 3-118
 Edit Changed Definitions (META-X), 3-16
 Edit Combined Methods (META-X), 3-121
 Edit Compiler Warnings (META-X), 3-30
 Edit Definition (META-.), 3-119
 Edit File Warnings (META-X), 3-30
 Edit Flavor Components (META-X), 3-121
 Edit Flavor Dependents (META-X), 3-121
 Edit Flavor Direct Dependents (META-X), 3-121
 Edit Flavor Methods (META-X), 3-121

Edit Methods (META-X), 3-121
 Edit Next Warning (CTRL-SHIFT-W), 3-31
 Edit Previous Warning (META-SHIFT-W), 3-31
 Edit System Warnings (META-X), 3-30
 Edit Tab Stops (META-X), 3-194
 Edit Warnings (META-X), 3-30
 Edit Word Abbrevs (META-X), 3-56
 Edit Zmacs Command (META-CTRL-.), 3-120
 Electric Font Lock Mode (META-X), 3-104, 3-145
 Electric Shift Lock Mode (META-X), 3-145
 End Kbd Macro (CTRL-X), 3-50
 End of Definition (META-CTRL-] or META-CTRL-↓), 3-38
 End of Line (CTRL-E or SUPER→), 3-34
 Evaluate and Exit (META-CTRL-Z), 3-26
 Evaluate and Replace Into Buffer (META-X), 3-7, 3-23
 Evaluate Buffer (META-X), 3-27
 Evaluate Buffer Changed Definitions (META-X), 3-26
 Evaluate Changed Definitions (META-X), 3-26
 Evaluate Into Buffer (META-X), 3-8, 3-23
 Evaluate Minibuffer (META-ESCAPE), 3-23, 3-114
 Evaluate Region (CTRL-SHIFT-E), 3-24
 Evaluate Region Hack (META-CTRL-SHIFT-E), 3-24
 Evaluate Region Verbose (META-SHIFT-E), 3-24
 Exchange Characters (CTRL-T), 3-180
 Exchange Lines (CTRL-X CTRL-T), 3-180
 Exchange Regions (CTRL-X T), 3-180
 Exchange Sexps (META-CTRL-T), 3-181
 Exchange Words (META-T), 3-181
 Execute Command Into Buffer (META-X), 3-8
 Expand Only (META-CTRL-space bar in Word Abbreviation mode; META-CTRL-X Expand Only otherwise), 3-56
 Expunge Directory (META-X), 3-76
 Extended Reverse String Search (META-X), 3-164
 Extended String Search (META-X), 3-164

F

Fill Long Comment (META-X), 3-134, 3-183
 Fill Paragraph (META-Q), 3-183
 Fill Region (META-G), 3-184
 Find File (CTRL-X CTRL-F), 3-96
 Find File No Sectionize (META-X), 3-96
 Find Pattern (META-X), 3-166
 Find Source Filename (META-), 3-120
 Find System Files (META-X), 3-96
 Find Unbalanced Parentheses (CTRL-(), 3-138

Finish Patch (META-X), 3-126
 Finish Patch Unreleased (META-X), 3-126
 Fontify Region or Buffer (META-X), 3-104
 Forward Character (CTRL-F or →), 3-34
 Forward List (META-CTRL-N), 3-38
 Forward Paragraph (META-]), 3-40
 Forward Sentence (META-E or META-↓), 3-41
 Forward Sexp (META-CTRL-F or META-CTRL-→), 3-38
 Forward Sexp No Up (META-X), 3-39
 Forward Up List (META-CTRL-), 3-39
 Forward Word (META-F or META-→), 3-41
 Function Apropos (META-X), 3-109
 Fundamental Mode (META-X), 3-144

G

Get Register (META-X), 3-71
 Go to Next Possibility (CTRL-SHIFT-P), 3-127
 Go to Next Top Level Possibility (META-X), 3-128
 Go to Possibility (CTRL-/), 3-127
 Go to Warning (CTRL-/), 3-31
 Goto Beginning (META-< or HYPER-↑), 3-34, 3-155
 Goto End (META-> or HYPER-↓), 3-34, 3-155
 Grind Definition (META-X), 3-136, 3-153
 Grind Expression (META-X), 3-136, 3-153
 Grow List Backward (META-X), 3-138
 Grow List Forward (META-X), 3-138
 Grow Window (CTRL-X ^), 3-197

I

Incremental Search (CTRL-S), 3-161
 Indent Comment Relative (META-X), 3-134, 3-189
 Indent Differently (CTRL-TAB), 3-190
 Indent for Comment (CTRL-; or META-;), 3-134, 3-190
 Indent for Lisp (TAB), 3-190
 Indent Nested (META-X), 3-190
 Indent New Comment Line (META-LINE FEED), 3-135, 3-191
 Indent New Line (LINE FEED), 3-191
 Indent New Line at Previous Sexp (META-X), 3-191
 Indent Region (META-CTRL-\), 3-191
 Indent Relative (META-X), 3-192
 Indent Rigidly (CTRL-X CTRL-I), 3-192
 Indent Sexp (META-CTRL-Q), 3-192
 Indent Under (META-X), 3-192
 Insert Buffer (META-X), 3-8
 Insert CR (RETURN), 3-186
 Insert Date (META-X), 3-142
 Insert FF (META-CLEAR SCREEN), 3-186
 Insert File (META-X), 3-97
 Insert File No Fonts (META-X), 3-97

Insert File Warnings (META-X), 3-31
 Insert Tab (TAB or META-TAB [depends on mode]), 3-194
 Insert Warnings (META-X), 3-31
 Insert Word Abbrevs (META-X), 3-56
 Install Command (META-X), 3-47
 Install Macro (META-X), 3-50
 Install Mouse Macro (META-X), 3-50

J

Jump to Saved Position (CTRL-X J), 3-42, 3-71
 Just One Space (META-X), 3-196

K

Kbd Macro Query (CTRL-X Q), 3-51
 Kill All Word Abbrevs (META-X), 3-56
 Kill Backward Up List (META-X), 3-63
 Kill Buffer (CTRL-X K), 3-8
 Kill Comment (META-CTRL-;), 3-135
 Kill Global Word Abbrev (META-X), 3-56
 Kill Line (CTRL-K), 3-63
 Kill Line Backward (CLEAR INPUT), 3-63
 Kill Local Variable (META-X), 3-54
 Kill Mode Word Abbrev (META-X), 3-56
 Kill or Save Buffers (META-X), 3-9
 Kill Region (CTRL-W), 3-64
 Kill Register (META-X), 3-72
 Kill Sentence (META-K), 3-64
 Kill Sexp (META-CTRL-K), 3-64
 Kill Sexp No Up (META-X), 3-64
 Kill Some Buffers (META-X), 3-10
 Kill Word (META-D), 3-64
 Kill Yank (Mouse-M2), 3-151

L

Lisp Lowercase Region (META-X), 3-184
 Lisp Match Search (CTRL-SHIFT-S), 3-167
 Lisp Uppercase Region (META-X), 3-184
 List All Directory Names (META-X), 3-76
 List Buffer Changed Definitions (META-X), 3-17
 List Buffers (CTRL-X CTRL-B), 3-13
 List Callers (META-X), 3-118
 List Changed Definitions (META-X), 3-16
 List Combined Methods (META-X), 3-121
 List Commands (META-X), 3-107
 List Files (META-X), 3-76
 List Flavor Components (META-X), 3-122
 List Flavor Dependents (META-X), 3-122
 List Flavor Direct Dependents (META-X), 3-122
 List Flavor Methods (META-X), 3-122
 List Flavor Methods All (META-X), 3-122
 List Fonts (META-X), 3-102
 List Keystroke History (HELP L), 3-114
 List Local Variables (META-X), 3-54

List Matching Lines (META-X), 3-167
 List Matching Symbols (META-X), 3-168
 List Methods (META-X), 3-122
 List Modifications (META-X), 3-18
 List Registers (META-X), 3-68
 List Sections (META-X), 3-17
 List Some Word Abbrevs (META-X), 3-57
 List Tag Tables (META-X), 3-172
 List Variables (META-X), 3-52
 List Word Abbrevs (META-X), 3-57
 Load File (META-X), 3-28
 Load Kbd Macro (META-X), 3-51
 Long Documentation (META-SHIFT-D),
 3-111
 Lowercase Region (CTRL-X CTRL-L), 3-184
 Lowercase Word (META-L), 3-185

M

Macro Expand Expression (CTRL-SHIFT-M),
 3-136
 Macro Expand Expression All
 (META-SHIFT-M), 3-136
 Macsyma Mode (META-X), 3-144
 Make () (META-(), 3-138
 Make () Backward (META-X), 3-139
 Make Local Variable (META-X), 3-54
 Make Room (CTRL-O), 3-186
 Make Word Abbrev (META-X), 3-57
 Mark Beginning (CTRL-<), 3-66
 Mark Definition (META-CTRL-H), 3-66
 Mark End (CTRL->), 3-66
 Mark Item (Mouse-M), 3-151
 Mark Page (CTRL-X CTRL-P), 3-66
 Mark Paragraph (META-H), 3-66
 Mark Region (Mouse-LHOLD), 3-152
 Mark Sexp (META-CTRL-@), 3-66
 Mark Whole (CTRL-X H), 3-67
 Mark Word (META-@), 3-67
 Modified Two Windows (CTRL-X 4), 3-197
 Move Over) (META-), 3-39, 3-192
 Move Point (Mouse-L), 3-152
 Move to Bottom of Screen (SUPER-↓), 3-34
 Move to Default Previous Point (CTRL-X
 META-CTRL-space bar), 3-42
 Move to Point (Mouse-L2), 3-152
 Move to Previous Point (META-CTRL-space
 bar), 3-42
 Move to Screen Edge (META-R), 3-35
 Move to Top of Screen (SUPER-↑), 3-35
 Multiple Edit Callers (META-X), 3-118
 Multiple List Callers (META-X), 3-119
 Multiple Query Replace (META-X), 3-170
 Multiple Query Replace From Buffer
 (META-X), 3-170

N

Name Last Kbd Macro (META-X), 3-50
 Next File (META-X), 3-172

Next Page (CTRL-X]), 3-155
 Next Screen (CTRL-V or CTRL-↓), 3-155
 Next Several Screens (META-X), 3-156
 Not Modified (META--), 3-18

O

Occur (META-X), 3-168
 One Window (CTRL-X 1), 3-198
 Open Get Register (CTRL-X G), 3-72
 Other Window (CTRL-X O or Mouse-L),
 3-152, 3-198
 Overwrite Mode (META-X), 3-146

P

Pop Minibuffer History (META-CTRL-Y in the
 minibuffer), 3-73, 3-114
 Prepend to File (META-X), 3-98
 Previous Page (CTRL-X [), 3-156
 Previous Screen (META-V or CTRL-↑), 3-156
 Previous Several Screens (META-X), 3-156
 Print All Buffers (META-X), 3-19, 3-153
 Print Buffer (META-X), 3-19, 3-153
 Print File (META-X), 3-98, 3-154
 Print Region (META-X), 3-154
 Push Pop Point Explicit (META-space bar),
 3-42
 Put Register (CTRL-X X), 3-68

Q

Query Exchange (META-X), 3-170
 Query Replace (META-%), 3-170
 Quick Arglist (CTRL-SHIFT-A), 3-111
 Quick Disassemble (META-X), 3-32
 Quick Documentation (CTRL-SHIFT-D),
 3-111
 Quick Print Buffer (META-SHIFT-P), 3-19,
 3-154
 Quick Redo (CTRL-SHIFT-R), 3-115
 Quick Undo (CTRL-SHIFT-U), 3-116
 Quit (END), 3-106

R

Read Word Abbrev File (META-X), 3-57
 Reap File (META-X), 3-95
 Recenter Window (CTRL-L), 3-198
 Redo (META-X), 3-116
 Release Patch (META-X), 3-125
 Rename Buffer (META-X), 3-18
 Rename File (META-X), 3-98
 Reparse Attribute List (META-X), 3-148
 Repeat Minibuffer Command (CTRL-X
 ESCAPE), 3-114
 Replace String (CTRL-%), 3-171
 Reposition Window (META-CTRL-R), 3-156,
 3-199
 Resume Patch (META-X), 3-126
 RETURN Indents Mode (META-X), 3-146

Reverse Following List (META-X), 3-182
 Reverse Incremental Search (CTRL-R), 3-162
 Reverse Lines (META-X), 3-182
 Reverse String Search (META-X or CTRL-R ESCAPE), 3-161
 Revert Buffer (CTRL-X CTRL-R), 3-20
 Right Adjust Line (META-X), 3-186

S

Save All Files (META-X), 3-99
 Save File (CTRL-X CTRL-S), 3-99
 Save Position (CTRL-X S), 3-42, 3-68
 Save Region (META-W), 3-65
 Save Word Abbrev File (META-X), 3-57
 Scroll Other Window (META-CTRL-V), 3-157, 3-199
 Sectionize Buffer (META-X), 3-18
 Select All Buffers as Tag Table (META-X), 3-172
 Select Buffer (CTRL-X B), 3-20
 Select Default Previous Buffer (CTRL-X META-CTRL-L), 3-20
 Select Previous Buffer (META-CTRL-L), 3-20
 Select System as Tag Table (META-X), 3-173
 Select Tag Table (META-X), 3-173
 Self Document (HELP C), 3-108
 Set Backspace (META-X), 3-148
 Set Base (META-X), 3-148
 Set Comment Column (CTRL-X ;), 3-135
 Set Default Filename (META-X), 3-98
 Set Fill Column (CTRL-X F), 3-183, 3-186
 Set Fill Prefix (CTRL-X .), 3-184
 Set Fonts (META-X), 3-102, 3-149
 Set Goal Column (CTRL-X CTRL-N), 3-187
 Set Key (META-X), 3-47
 Set Lowercase (META-X), 3-149
 Set Nofill (META-X), 3-149
 Set Package (META-X), 3-123, 3-149
 Set Patch File (META-X), 3-150
 Set Pop Mark (CTRL-space bar), 3-43, 3-67
 Set Tab Width (META-X), 3-150, 3-195
 Set Variable (META-X), 3-53
 Set Version Limit (META-X), 3-95
 Set Visited Filename (META-X), 3-98
 Set VSP (META-X), 3-150
 Show All Registers (META-X), 3-69
 Show Editor Menu (Mouse-R), 3-152
 Show Kill History (CTRL-STATUS), 3-73
 Show List Start (META-X), 3-139
 Show Point PDL (META-STATUS), 3-43
 Show Saved Positions (META-X), 3-44, 3-70
 Show System Menu (Mouse-R2), 3-152
 Simple Exchange Characters (META-X), 3-181
 Sort Decreasing Creation Date (META-X), 3-90
 Sort Decreasing Filename (META-X), 3-91
 Sort Decreasing Reference Date (META-X), 3-91

Sort Decreasing Size (META-X), 3-91
 Sort Increasing Creation Date (META-X), 3-91
 Sort Increasing Filename (META-X), 3-91
 Sort Increasing Reference Date (META-X), 3-91
 Sort Increasing Size (META-X), 3-91
 Sort Lines (META-X), 3-187
 Sort Paragraphs (META-X), 3-187
 Sort via Keyboard Macros (META-X), 3-187
 Source Compare (META-X), 3-128
 Source Compare Changes (META-X), 3-128
 Source Compare Merge (META-X), 3-128
 Split Line (META-CTRL-O), 3-187
 Split Screen (META-X), 3-199
 Stack List Vertically (META-X), 3-193
 Start Kbd Macro (CTRL-X (), 3-50
 Start Patch (META-X), 3-124
 Start Private Patch (META-X), 3-124
 String Search (META-X or CTRL-S ESCAPE), 3-159
 Stupid Tab (TAB [only in Fundamental mode]), 3-195
 Swap Point and Mark (CTRL-X CTRL-X), 3-44, 3-67
 Symbol Help (HELP S), 3-115

T

Tab Hacking Delete Forward (META-X), 3-61
 Tab Hacking Rubout (RUBOUT in Common Lisp or Zetalisp mode; CTRL-RUBOUT otherwise), 3-61
 Tab to Tab Stop (TAB in Text mode; META-X otherwise), 3-195
 Tabify (META-X), 3-195
 Tags Compile Changed Definitions (META-X), 3-173
 Tags Edit Changed Definitions (META-X), 3-173
 Tags Evaluate Changed Definitions (META-X), 3-174
 Tags List Changed Definitions (META-X), 3-174
 Tags Multiple Query Replace (META-X), 3-174
 Tags Multiple Query Replace From Buffer (META-X), 3-174
 Tags Query Replace (META-X), 3-175
 Tags Search (META-X), 3-175
 Tags Search List Definitions (META-X), 3-175
 Tags Search Next Occurrence (META-X), 3-175
 Teach Zmacs (META-X), 3-115
 Text Mode (META-X), 3-143
 This Indentation (META-O), 3-193
 Toggle Read Only (CTRL-X R), 3-18
 Trace (META-X), 3-132

Two Windows (CTRL-X 2), 3-201
Two Windows Showing Region (CTRL-X 8),
3-201

U

Uncomment Out Region (CTRL-X META-;),
3-133
Uncomment Region (META-X), 3-135
Undelete File (META-X), 3-95
Undo (HELP U or UNDO), 3-116
Unexpand Last Word (CTRL-X U), 3-57
Untabify (META-X), 3-195
Up Comment Line (META-P), 3-39, 3-135
Up Indented Line (META-X), 3-39
Up Real Line (CTRL-P or ↑), 3-35
Update Attribute List (META-X), 3-148
Update Xld (CTRL-X
META-CTRL-SHIFT-C), 3-28
Uppercase Global Functions Mode (META-X),
3-146
Uppercase Global Functions Region
(META-X), 3-147
Uppercase Initial (META-C), 3-185
Uppercase Previous Global Function
(META-X), 3-146
Uppercase Region (CTRL-X CTRL-U), 3-185
Uppercase Word (META-U), 3-185

V

Variable Apropos (HELP V), 3-52, 3-112
Various Quantities (CTRL-Q), 3-45, 3-142
View Buffer (CTRL-X V), 3-19

View Directory (META-X), 3-76
View File (META-X), 3-96
View Kbd Macro (META-X), 3-51
View Login Directory (META-X), 3-76
View Register (META-X), 3-70
View Two Windows (CTRL-X 3), 3-201
Visit File (CTRL-X CTRL-V), 3-97
Visit Tag Table (META-X), 3-176

W

Where Am I (CTRL=), 3-115
Where Is (HELP W), 3-108
Where Is Symbol (META-X), 3-112, 3-168
Word Abbrev Mode (META-X), 3-57, 3-146
Word Abbrev Prefix Mark (META-X), 3-58
Write File (CTRL-X CTRL-W), 3-99
Write Kbd Macro (META-X), 3-51
Write Region (META-X), 3-99
Write Word Abbrev File (META-X), 3-58

Y

Yank (CTRL-Y), 3-73
Yank Default String (CTRL-SHIFT-Y in the
minibuffer), 3-114
Yank Pop (META-Y [executed immediately
after a CTRL-Y or META-Y]), 3-73
Yank Search String (CTRL-S CTRL-S), 3-163

Z

Zetalisp Mode (META-X), 3-143
Ztop Mode (META-X), 3-144

Data Systems Group – Austin Documentation Questionnaire

Explorer Zmacs Editor Reference

Do you use other TI manuals? If so, which one(s)?

_____	_____
_____	_____
_____	_____

How would you rate the quality of our manuals?

	Excellent	Good	Fair	Poor
Accuracy	_____	_____	_____	_____
Organization	_____	_____	_____	_____
Clarity	_____	_____	_____	_____
Completeness	_____	_____	_____	_____
Overall design	_____	_____	_____	_____
Size	_____	_____	_____	_____
Illustrations	_____	_____	_____	_____
Examples	_____	_____	_____	_____
Index	_____	_____	_____	_____
Binding method	_____	_____	_____	_____

Was the quality of documentation a criterion in your selection of hardware or software?

- Yes No

How do you find the technical level of our manuals?

- Written for a more experienced user than yourself
 Written for a user with the same experience
 Written for a less experienced user than yourself

What is your experience using computers?

- Less than 1 year 1-5 years 5-10 years Over 10 years

We appreciate your taking the time to complete this questionnaire. If you have additional comments about the quality of our manuals, please write them in the space below. Please be specific.

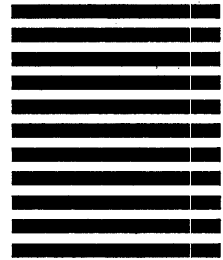
Name _____ Title/Occupation _____
Company Name _____
Address _____ City/State/Zip _____
Telephone _____ Date _____

TAPE EDGE TO SEAL

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP

ATTN: PUBLISHING CENTER
P.O. Box 2909 M/S 2146
Austin, Texas 78769-9990



FOLD