

3 Text Editing and Processing

symbolics



3 Text Editing and Processing

symbolics[™]

Text Editing and Processing

999020

July 1986

This document corresponds to Genera 7.0 and later releases.

The software, data, and information contained herein are proprietary to, and comprise valuable trade secrets of, Symbolics, Inc. They are given in confidence by Symbolics pursuant to a written license agreement, and may be used, copied, transmitted, and stored only in accordance with the terms of such license. This document may not be reproduced in whole or in part without the prior written consent of Symbolics, Inc.

Copyright © 1986, 1985, 1984, 1983, 1982, 1981, 1980 Symbolics, Inc. All Rights Reserved.

Portions of font library Copyright © 1984 Bitstream Inc. All Rights Reserved.

Portions Copyright © 1980 Massachusetts Institute of Technology. All Rights Reserved.

Symbolics, Symbolics 3600, Symbolics 3670, Symbolics 3675, Symbolics 3640, Symbolics 3645, Symbolics 3610, Genera, Symbolics-Lisp[®], Wheels, Symbolics Common Lisp, Zetalisp[®], Dynamic Windows, Document Examiner, Showcase, SmartStore, SemantiCue, Frame-Up, Firewall, S-DYNAMICS[®], S-GEOMETRY, S-PAINT, S-RENDER[®], MACSYMA, COMMON LISP MACSYMA, CL-MACSYMA, LISP MACHINE MACSYMA, MACSYMA Newsletter and Your Next Step In Computing are trademarks of Symbolics, Inc.

Restricted Rights Legend

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

Symbolics, Inc.
4 New England Tech Center
555 Virginia Road
Concord, MA 01742

Text written and produced on Symbolics 3600-family computers by the Documentation Group of Symbolics, Inc.

Text masters produced on Symbolics 3600-family computers and printed on Symbolics LGP2 Laser Graphics Printers.

Cover design: Schafer/LaCasse

Printer: CSA Press

Printed in the United States of America.

Printing year and number: 88 87 86 9 8 7 6 5 4 3 2 1

Table of Contents

	Page
I. Zmacs Manual	1
1. Introduction to the Zmacs Manual	3
Overview of the Zmacs Manual	4
Introduction to Zmacs	6
Zmacs Manual Notation Conventions	9
2. Getting Started in Zmacs	11
Entering Zmacs	12
Zmacs Help	14
Organization of the Screen	18
Inserting Text	24
Numeric Arguments	26
Introduction to Moving the Cursor	28
Introduction to Erasing Text	30
Creating and Saving Buffers and Files	32
Zmacs Commands for Formatting Text	35
Executing CP Commands From Zmacs	43
Leaving Zmacs	44
3. Getting Help in Zmacs	47
Getting Out of Trouble	48
Finding Out About Zmacs Commands	51
The Editor Menu	57
More on the Minibuffer	59
Getting Information About Buffers and Regions	61
4. Moving the Cursor in Zmacs	63
Overview of Moving the Cursor	64
Redisplaying the Window	65
Moving the Cursor with the Mouse	67
Motion Commands	71
Motion by Lisp Expression	76
Motion by Paragraph	79
Motion by Page	80

Motion with Respect to the Whole Buffer	81
5. Deleting and Transposing Text in Zmacs	83
Deleting Vs. Killing Text	84
Deleting and Transposing Characters	90
Deleting and Transposing Words	92
Deleting and Transposing Lisp Expressions	93
Deleting and Transposing Lines	95
Deleting Sentences	97
6. Working with Regions in Zmacs	99
What is a Zmacs Region?	100
Registers in Zmacs	104
Commands to Mark Regions	106
Region-Manipulating Commands	109
7. Searching, Replacing, and Sorting in Zmacs	113
Searching in Zmacs	114
Locating and Replacing Strings Automatically	118
Tag Tables and Search Domains	122
Sorting	128
8. Manipulating Buffers and Files in Zmacs	129
Working with Buffers and Files	130
Selecting, Listing, and Examining Buffers	132
Buffer Commands	133
Appending, Prepending, and Inserting Text	141
Comparing Files and Buffers	142
Window Commands	146
File Manipulation Commands	148
Buffer and File Attributes	155
Dired Mode	163
9. Setting the Zmacs Major Mode	175
Major Editing Modes	176
10. Zmacs Speller	179
Using the Zmacs Speller	180
Speller Commands for Spelling	184
Speller Dictionaries	187

11. Word Abbreviations	197
Using Word Abbreviations	198
Word Abbreviation Commands	200
12. Using Character Styles in Zmacs	205
Introduction to the Character Style Commands	206
Character Style Commands in Zmacs	210
13. Changing Case and Indentation in Zmacs	213
Changing Case	214
Indentation	216
14. Editing Lisp Programs in Zmacs	223
Introduction	224
Commenting Lisp Code	226
Evaluating and Compiling Lisp Programs	229
Parenthesizing Lisp Expressions	233
Expanding Lisp Expressions	234
Locating Source Code to Edit	235
Patching	242
15. Customizing the Zmacs Environment	251
Overview	252
Built-in Customization Using Zmacs Minor Modes	253
Major Modes	256
Creating New Commands with Keyboard Macros	257
Key Bindings	267
How to Specify Zmacs Variable Settings	269
Customizing Zmacs in Init Files	272
Appendix A. Zmacs Help Command Summary	277
Zmacs Commands for Finding Out About the State of Buffers	278
Zmacs Commands for Finding Out About the State of Zmacs	279
Zmacs Commands for Finding Out About Lisp	280
Zmacs Commands for Finding Out About Flavors	281
Zmacs Commands for Interacting with Lisp	282
II. Font Editor	283
16. Font Basic Concepts	285

16.1	Attributes of TV Fonts	285
16.2	Standard TV Fonts	287
17.	Entering and Leaving FED	289
18.	Font Editor Basic Concepts	291
18.1	FED, the Subsystem	291
18.2	Selecting a Font	294
18.2.1	Creating a New Font	297
18.2.2	Displaying Characters in the Font	297
18.3	Selecting a Character	297
18.3.1	From the Character Select Menu	298
18.3.2	By Creating a New Character	298
18.3.3	From the [Show Font] Display	298
18.3.4	With the C Command	298
18.3.5	By Renaming Characters	298
19.	Drawing	299
19.1	Drawing Characters with the Mouse	299
19.2	The Nonmouse Cursor	300
20.	Viewing and Altering a Character in the Character Box	301
20.1	What the Lines Mean	301
20.2	Altering the Character Box	302
21.	The Gray Plane	303
21.1	Getting Things Into Gray	303
21.1.1	With [Swap Gray]	303
21.1.2	With [Gray Char]	303
21.2	Merging Characters with the Gray Plane	304
22.	Saving Characters and Pieces of Characters in Registers	307
22.1	Saving a Drawing Into a Register	307
22.2	Retrieving the Contents of a Register	307
22.3	Retrieving the Black Plane While Manipulating Registers	307
23.	Transformations	309
23.1	Clearing the Drawing	309
23.2	Rotating Drawings	309
23.3	Reflecting Drawings	309
23.4	Moving the Drawing	312

23.5	Drawing Lines and Curves	312
23.6	Stretching and Contracting	316
23.6.1	Stretching a Drawing Horizontally	316
23.6.2	Contracting a Drawing Horizontally	316
23.6.3	Stretching a Drawing Vertically	316
23.6.4	Contracting a Drawing Vertically	316
24.	The Sample String	321
25.	Adjusting the Display	323
25.1	Positioning the Drawing	323
25.2	Setting the Box Size in the Drawing Pane	324
25.3	Setting the Height and Width of the Drawing Pane	324
26.	Reading and Writing Files	325
26.1	Reading Files	325
26.2	Writing Files	326
27.	Command List	327
27.1	Menu and Keyboard Commands	327
27.1.1	Configuration and Drawing Transformation	327
27.1.2	Gray Plane Menu Items	328
27.1.3	Outside World Interface Menu Items	329
27.1.4	Evaluating Forms From FED	330
27.2	Keyboard-only Commands	330
27.3	Mouse Sensitivities	330
27.3.1	The Drawing Pane	330
27.3.2	The Draw Mode Menu	331
27.3.3	The Sample Pane	331
27.3.4	The Character Select Pane	331
27.3.5	The Font Parameters Menu	331
27.3.6	The Register Pane	331
27.3.7	The List Fonts and Show Font Displays	332
Index		333

List of Figures

Figure 1.	Initial FED Display	290
Figure 2.	Tall Configuration	295
Figure 3.	Wide Configuration	296
Figure 4.	[Rotate (R)]	310
Figure 5.	[Rotate (M)]	311
Figure 6.	[Reflect]	313
Figure 7.	The X axis (-)	314
Figure 8.	Moving the Drawing with [Move Black]	315
Figure 9.	Stretching Horizontally	317
Figure 10.	Contracting Horizontally	318

PART I.

Zmacs Manual

1. Introduction to the Zmacs Manual

Overview of the Zmacs Manual

Scope

The *Zmacs Manual* is primarily a reference manual and is intended for all users of Zmacs on the Lisp Machine. It contains both conceptual overview and reference material that together describe the Zmacs editor. We assume that you have already read the *User's Guide to Symbolics Computers*.

Organization

The first three chapters contain introductory material for users who are unfamiliar with Zmacs concepts. Experienced users can skim the remaining chapters, which are organized according to editing function, and use them as reference material.

"Introduction" gives an overview of Zmacs and describes Zmacs documentation conventions in this manual.

"Getting Started" introduces basic Zmacs concepts and commands, such as how to enter text, move the cursor, and make simple corrections.

"Getting Help" describes ways to get out of trouble and how to get Zmacs information during editing.

"Moving the Cursor" includes descriptions of both mouse and keyboard motion commands.

"Deleting and Transposing Text" explains Zmacs deletion and text retrieval concepts, as well as the ways to delete and transpose text.

"Working With Regions" tells how to manipulate blocks of text.

"Searching, Replacing, and Sorting" describes the commands for locating and replacing character strings in one or many files.

"Manipulating Buffers and Files" gives more information on manipulating blocks of text, inserting files, keeping track of everything, and editing your directory.

"Setting the Major Mode" documents the major editing modes and their characteristics.

"Changing Case and Indentation" includes many commands for changing code, comments, or text to uppercase or lowercase, as well as commands for handling white space, indentation, and formatting.

"Editing Lisp Programs" the ways in which Zmacs is tailored for use in writing and editing programs in Lisp.

"Customizing the Zmacs Environment" describes how to fine tune

Overview of the Zmacs Manual, *cont'd.*

your Zmacs environment using modes to set it up, keyboard macros to perform special editing tasks, binding keys to the commands of your choice, setting Zmacs variables to alter your standard system aults, and saving the customized environment in init files.

Appendix A summarizes Zmacs help commands according to the context in which they are available.

Introduction to Zmacs

Overview

Zmacs, the Lisp Machine editor, is built on a large and powerful system of text-manipulation functions and data structures, called *Zwei*.

Zwei is not an editor itself, but rather a system on which other text editors are implemented. For example, in addition to Zmacs, the Zmail mail reading system also uses Zwei functions to allow editing of a mail message as it is being composed or after it has been received. The subsystems that are established upon Zwei are:

- Zmacs, the editor that manipulates text in files
- Dired, the editor that manipulates directories represented as text in files
- Zmail, the editor that manipulates text in mailboxes
- Converse, the editor that manipulates text in messages

Since these subsystems share Zwei in the dynamically linked Lisp environment, many of the commands available as Zmacs commands are available in other editing contexts as well.

In this manual, we discuss Zmacs commands in the context of Zmacs only. We also describe Dired, the directory editor, since it is used within Zmacs.

Commands

Zmacs *commands* are Lisp functions that perform the editing work. Every Zmacs command has a *name*, and many commands are bound to keys. When a command is bound to a *keystroke combination*, you invoke it by pressing those keys. For example, the Forward Word command is invoked by typing the keystroke `m-F`. When a command is not bound to a set of keystrokes, Zmacs calls it an *extended* command and you invoke it using its name preceded by `m-X`. For example, the command View Mail, an extended command, is invoked by View Mail `m-X`.

Command tables assign keystrokes and names to commands. Each time you press a key, Zmacs looks up the function associated with that key. For ordinary characters, the function **com-standard**, in the standard command table, inserts the character once.

Introduction to Zmacs, *cont'd.*

Keystrokes

A keystroke has a character component and a modifier component, and is performed by pressing a *primary key* (alphanumeric), possibly while holding down a *shift key* or a group of shift keys. The primary key held down with either the SHIFT or SYMBOL keys determines the *character* part of a keystroke. Whether you hold down the other shift keys, CONTROL, META, HYPER, and SUPER, determines the *modifier* part of the keystroke.

In general, commands that begin with a CONTROL (c-) key modifier operate on single characters, commands that begin with a META (m-) key modifier operate on words, sentences, paragraphs, and regions, and commands that begin with a CONTROL META (c-m-) modifier operate on Lisp code.

Prefix character commands consist of more than one keystroke per command. For example, to invoke the command c-X F, you first type the prefix character c-X and then the primary key F. Prefix character commands are not case-sensitive – that is, Zmacs converts a lowercase character following a prefix character command (like c-X) to uppercase. For example, c-X f is equivalent to c-X F.

Zmacs commands are self-delimiting. Unless otherwise specified, you do not need to type a carriage return or other terminating character to finish typing a command.

Extended Commands

Extended commands extend the range of commands past the one-or-two-keystroke limitation. You invoke Zmacs extended commands by name using the m-X command:

m-X Extended Command

Prompts for the name of a Zmacs command and executes that command.

Command completion is provided. See the section "Completion for Extended Commands (m-X Commands)" in *User's Guide to Symbolics Computers*.

Command Tables

There is always a currently active command table (*comtab*). When you invoke a command, Zmacs looks it up in the associated command table, checks to see if it is valid in the current context, and performs the function. Zmacs uses many comtabs, including the standard comtab, a comtab for commands that begin with the c-X prefix, and a comtab for reading pathnames in the minibuffer.

Introduction to Zmacs, *cont'd.*

Many commands have no meaning outside their own limited context. Sometimes you might get a message or see online documentation about a command that says `Not available in current context.` Those commands that are not accessible via a keystroke and not accessible via `m-X` are likely to be commands that do not work in the current context. For example, a command that is part of `Dired` is only available on a key when you are in `Dired`.

You can invoke a command that is not available in the current comtab with the `c-m-X` command. `c-m-X` works like `m-X`: you press the keys and then type the command name in the minibuffer. This is primarily intended for debugging new editor commands that have not yet been installed on any key. Using `c-m-X` to invoke a command that is not in the current comtab because it works only in some other context is a sure way to get into trouble.

`c-m-X`

Any Extended Command

Prompts for the name of a Zmacs command and executes that command.

Command completion is provided.

Zmacs Manual Notation Conventions

Zmacs Notation

Conventions and Examples

The word *current*, when describing a word, line, paragraph, page, or any Zmacs-recognizable piece of text, refers to the text that currently contains (or immediately follows) the cursor.

The *invocation* of a command shows exactly what keys you must press to invoke, or call, a command. We use the following format to describe Zmacs commands:

```
invocation                               Name
alternate invocation
alternate invocation
```

Formal description of command

Since each extended ($m-X$) command contains its name as part of its invocation, we do not repeat the name again on that line.

Example 1 of Zmacs Notation Conventions

```
m->                                     Goto End
```

Moves point to the end of the buffer.

With a numeric argument n between 0 and 10, moves point to a place $n/10$ of the way from the end of the buffer to the beginning.

(The $m->$ command goes to the end of the buffer – its name is Goto End.)

Example 2 of Zmacs Notation Conventions

```
Dired (m-X)
```

Prompts for the name of a directory to edit with Dired.

(The Dired ($m-X$) command is an extended command that enters the directory editor.)

Zmacs Manual Notation Conventions, *cont'd.*

Example 3 of Zmacs

Notation Conventions

m-M

Back To Indentation

c-m-M

m-RETURN

c-m-RETURN

Positions point before the first nonblank character on the current line.

(Back to Indentation has several possible invocations that all move back to the first nonblank character on the line.)

2. Getting Started in Zmacs

Entering Zmacs

Introduction to Entering Zmacs

You can enter, or invoke, the editor in several ways: Press SELECT E, use the mouse, or run either the function `ed` or the function `zwei:edit-functions`. You can also use the command Select Activity, specifying either Zmacs or Editor as its argument.

Entering Zmacs with SELECT E

You can invoke the editor by pressing the SELECT key and then the letter E:

- If you have already been in the editor since booting the machine, Zmacs returns you to the same place in the same buffer that you last used.
- If this is the first time you are entering Zmacs since booting the machine, Zmacs puts you in an empty buffer named `*Buffer-1*`.

SELECT E enters or returns you to the editor from anyplace in the system, not just when you are talking to Lisp.

Entering Zmacs with the Mouse

You can invoke the editor using the mouse.

Summon a System menu by clicking right twice [(R2)]. Then click left on the Edit option [Edit (L)], which puts you into a Zmacs buffer. As for SELECT E, if you are returning to the editor Zmacs puts you back at the same place in the same buffer, and if you are entering Zmacs for the first time it puts you in an empty buffer.

Entering Zmacs with ed

The Lisp function `ed` enters Zmacs from a Lisp Listener. See the function `ed` in *User's Guide to Symbolics Computers*.

When reentering Zmacs within a login session, `ed` enters the editor, preserving its state as it was when you left. When entering Zmacs for the first time during a login session, `ed` initializes Zmacs and creates an empty buffer.

`arg` can have these values.

Entering Zmacs, *cont'd.*

<i>Value</i>	<i>Description</i>
t	The ed function enters the editor, creates an empty buffer, and selects it.
Pathname or string <i>(ed "filename")</i>	The ed function enters the editor and finds or creates a buffer with the specified file in it.
Defined symbol	The editor tries to find the source definition of that symbol for you to edit. A defined symbol can be, for example, a function, macro, variable, flavor, or system.
The symbol zwei:reload	The system reinitializes the editor. This destroys all existing buffers, so use this only if you have to.

Entering Zmacs with **zwei:edit-functions**

The Lisp function **zwei:edit-functions** also enters Zmacs from a Lisp Listener.

zwei:edit-functions Function

zwei:edit-functions is like **ed** in that inside the editor process it throws you back into the editor, whereas from another process it just sends a message to the editor and selects the editor's window. **zwei:edit-functions** gives *spec-list* to the editor in the same way that Edit Callers and similar editor commands would. See the section "The Zmacs Edit Callers Commands", page 239.

This command is useful when you have collected the names of things that you need to change, for example, using some program to generate the list. *spec-list* is a list of definitions; these are either function specs (if the definitions are functions) or symbols.

Zmacs sorts the list into an appropriate order, putting definitions from the same file together, and creates a support buffer called ***Function-Specs-to-Edit-n***. It selects the editor buffer containing the first definition in the list.

Zmacs Help

Introduction to Zmacs Help

Zmacs has many features that provide information about Zmacs commands, existing code, buffers, and files. Two features are generally useful: the HELP key and completion. See the section "Getting Help in Zmacs", page 47.

Introduction to HELP

Pressing the HELP key in a Zmacs editing window gives information about Zmacs commands and variables. For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269. The kind of information it displays depends on the key you press after HELP.

HELP ?	Displays a summary of HELP options.
HELP A	Displays names, key bindings, and brief descriptions of commands if their names or the first lines of their help documentation contain a string you specify. The A refers to $m-X$ Apropos, the command equivalent. If you enter the command with a numeric argument, only the names of the commands are searched for the string, and not the help documentation.
HELP C	Displays the name and description of a command bound to a key you specify.
HELP D	Displays documentation for a command whose name you specify.
HELP L	Displays a listing of the last 60 keys you pressed.
HELP U	Offers to undo the last major Zmacs operation, such as sorting or filling, when possible.
HELP V	Displays the names and values of Zmacs variables whose names contain a string you specify. For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.
HELP W	Displays the key binding for a command you specify. (The W refers to where.)
HELP SPACE	Repeats the last HELP command.

Zmacs Help, *cont'd.*

Introduction to Completion

Some Zmacs operations require you to provide names – for example, names of extended commands, Lisp objects, buffers, or files. Often you do not have to type all the characters of a name; Zmacs offers *completion* over some names. When completion is available, the word Completion appears in parentheses above the right side of the minibuffer.

You can request completion when you have typed enough characters to specify a unique word or name. For extended commands and most other names, completion works on initial substrings of each word. For example, `m-X c SPACE b` is sufficient to specify the extended command Compile Buffer. `SPACE`, `COMPLETE`, `RETURN`, and `END` complete names in different ways. Press `HELP` or click right once, `[(R)]`, on the editor window or minibuffer to list possible completions for the characters you have typed. `c-/'` displays every command that contains the substring.

<code>SPACE</code>	Completes words up to the current word.
<code>HELP</code> or <code>c-?</code>	Displays possible completions in the typeout area.
<code>[(R)]</code>	Pops up a menu of possible completions.
<code>c-/'</code>	Runs Apropos for each of the partially typed words in the name.
<code>COMPLETE</code>	Completes as much as possible. This could be the full name.
<code>RETURN</code> or <code>END</code>	Confirms the name if possible, whether or not you have seen the full name.

Introduction to Yanking

Yanking is getting back previously killed text or previously typed minibuffer commands. Using a few keystrokes, you can retrieve editing commands you have previously typed or restore any piece of text you have previously killed. *Popping* is moving through the *history* of previous commands or killed text.

To get back the last piece of text killed, press `c-Y`. To work your way back through the *kill history*, press `m-Y` repeatedly.

To get back the last command typed, press `c-m-Y`. To work your

Zmacs Help, *cont'd.*

way back through the *command history*, press `m-Y` repeatedly. This technique works only on editor commands that use the minibuffer in some way. Commands like `c-N` or `c-P` are not kept in the history.

You can also select items from the entire kill history or command history, or search a part of these histories using string-matching.

Here are the commands for yanking in the kill history:

<code>c-Y</code>	Yanks the first element in the kill history.
<code>m-Y</code>	After <code>c-Y</code> , <code>m-Y</code> yanks the previous element in the kill history. Subsequent <code>m-Y</code> s move down the kill history.
<code>c-0 c-Y</code>	Displays the elements of the kill history. You can click left on any item in the history to insert it in your buffer. Only the first 25 elements of the history are shown. Click left on (<i>N</i> more elements in history.) to display the rest of the history.
<code>c-sh-Y</code>	Yanks the first element in the kill history that matches a string you supply.
<code>m-sh-Y</code>	After <code>c-sh-Y</code> , <code>m-sh-Y</code> yanks a previous element in the kill history that matches the string you supplied. Subsequent <code>m-sh-Y</code> s move down the kill history matching the same string. If pressed after a <code>c-Y</code> , <code>m-sh-Y</code> prompts for a string to match.
<code>c-0 c-sh-Y</code>	Displays the elements of the kill history that match a string you supply. You can click left on any item in the history to insert it in your buffer. Only the first 25 matching elements of the history are shown. Click left on (<i>N</i> more elements in history.) to display the rest of the matching history.

Here are the commands for yanking in the command history:

<code>c-m-Y</code>	Yanks the first element in the command history.
<code>m-Y</code>	After <code>c-m-Y</code> , yanks the previous element in the command history. Subsequent <code>m-Y</code> s move down the command history.

Zmacs Help, *cont'd.*

<code>c-@ c-m-Y</code>	Displays the elements of the command history. You can click left on any item in the history to execute that command. Only the first 25 elements in the history are shown. Click left on (<i>N</i> more elements in history.) to display the rest of the history.
<code>c-m-sh-Y</code>	Yanks the first element in the command history that matches a string you supply.
<code>m-sh-Y</code>	After <code>c-m-sh-Y</code> , <code>m-sh-Y</code> yanks a previous element in the command history that matches the string you supplied. Subsequent <code>m-sh-Y</code> s move down the command history matching the same string. If pressed after a <code>c-m-Y</code> , <code>m-sh-Y</code> prompts for a string to match.
<code>c-@ c-m-sh-Y</code>	Displays the elements of the command history that match a string you supply. You can click left on any item in the history to execute that command. Only the first 25 elements in the history are shown. Click left on (<i>N</i> more elements in history.) to display the rest of the history.

For complete descriptions of killing and yanking: See the section "Deleting and Transposing Text in Zmacs", page 83.

Organization of the Screen

Introduction to the Organization of the Screen

Zmacs divides its window into several areas: the editor window, the echo area, and the mode line, each of which contains its own type of information.

Zmacs Editor Window

The biggest area, the *editor window*, shows the text you are editing. You can edit several different items at once with Zmacs; each item is edited in a separate editing environment called a *buffer*.

Editor Window's Buffer

Zmacs gives every buffer a name. At any time you are editing only one of them, the *selected* buffer. When we speak of what some command does to "the buffer", we are talking about the currently selected buffer. Multiple buffers in Zmacs make it easy to switch among several files; the mode line tells you which one you are editing.

Editor Window's Cursor and Point

The small blinking rectangle, the *cursor*, usually appears somewhere within the buffer, showing the position of *point*, the location at which editing takes place. Although the cursor covers a single character, we consider point to be at the left edge of the cursor, between the character the cursor is blinking on and the previous character.

Editor Window's Typeout

When you request some other information from Zmacs (for example, if you ask for help or a listing of a file directory), Zmacs needs room to display this type of information. It prints this *typeout* in a *typeout window* (at the top of the editor window), which temporarily overlays the text in the editor window, using as much room as it needs.

Since the typeout is not part of the file you are editing, Zmacs delineates it from the editor buffer by drawing a line across the window between them (if both are present). The typeout window goes away if you type any command; if you want to make it go away immediately but not do anything else, you can press SPACE. The cursor, which appears at the end of the typeout, then moves back to its original location in the buffer.

Organization of the Screen, *cont'd.*

Zmacs Echo Area

A few lines at the bottom of the screen make up what is called the *echo area*. *Echoing* means displaying the commands that you type. Zmacs commands are usually not echoed at all, but if you pause in the middle of a multicharacter command, then all the characters (including numeric arguments and prefixes) typed so far are echoed. This is intended to prompt you for the rest of the command. The rest of the command is echoed, too, as you type it. This behavior is designed to give confident users optimum response, while giving hesitant users maximum feedback.

Echo Area's Minibuffer

Many Zmacs commands prompt you for additional information. This prompting happens in a small window within the echo area called the *minibuffer*.

When Zmacs prompts you, the cursor in the main editing window stops blinking and a blinking cursor appears in the minibuffer. Over the minibuffer, in the Zmacs mode line, some prompting text appears, indicating what information Zmacs is prompting you for.

When you type a response to the prompt, that response is inserted in the minibuffer. You can edit text in the minibuffer using the same Zmacs commands used in the main Zmacs window.

When you are done typing (and possibly editing) a response to the prompt, the RETURN key finishes your response.

Zmacs Mode Line

The line above the echo area is known as the *mode line*. It is the line that usually starts with ZMACS (Fundamental). Its purpose is to display information about the current buffer.

The mode line consists of:

- The name of the major mode
- The name of the minor mode(s), if any
- The name of the buffer
- The version number of the file
- The status of the buffer
- A message telling whether the buffer contents extend above and/or below the screen

The mode line has this format:

Organization of the Screen, *cont'd.*

ZMACS (major-mode minor-mode(s)) buffer (version) buffer-status
[position-flag]

Mode Line's Major-mode

major-mode is always the name of the *major mode* you are in. At any time, Zmacs is in one and only one of its possible major modes. The major modes available include:

- Fundamental mode (which Zmacs starts out in)
- Text mode
- Lisp mode
- MACSYMA mode

For full details about all the major modes, how they differ, and how to select one: See the section "Setting the Zmacs Major Mode", page 175.

Mode Line's Minor-mode

minor-mode is a list of the *minor modes* that are turned on at the moment. For example:

Fill	Auto Fill Mode
Electric Shift-lock	Electric Shift Lock Mode
Abbrev	Word Abbrev Mode
Overwrite	Overwrite Mode

For more information: See the section "Built-in Customization Using Zmacs Minor Modes", page 253.

Mode Line's Buffer

buffer is the name of the workspace that holds the text you are editing. A buffer can be named in one of two ways:

- By Zmacs, with a name that corresponds to the existing file that it contains or with its standard name for an empty buffer
- By you, with any name you like

When a buffer contains a file, the buffer name is the pathname of that file, rearranged with the file name first and the host and directory at the end. For a description of pathname components: See the section "Pathnames" in *Reference Guide to Streams, Files, and I/O*. When a buffer does not contain a file, the buffer name is a string.

Organization of the Screen, *cont'd.*

Buffers that do not contain files are empty, newly created, or temporary buffers. When Zmacs creates and names a buffer, that name begins and ends with an asterisk. When you create and name a buffer, on the other hand, its name is of your choosing.

When you first start up and enter Zmacs, your buffer is either:

- An empty buffer called *Buffer-1*, which is the only one that exists when Zmacs starts up
- A buffer containing an existing file, which Zmacs appropriately calls by its name

For information on multiple buffers: See the section "Manipulating Buffers and Files in Zmacs", page 129.

Mode Line's Version

(*version*) is the version number most recently visited or saved. The mode line does not display any version number if the file is on a file system that does not support version numbers, such as UNIX.

Mode Line's Buffer-status

If the mode line displays *, then changes have been made in the buffer that have not been saved in the file. If the buffer has not been changed since it was read in or saved, the mode line does not display a asterisk.

Mode Line's Position-flag

When the mode line displays the message [More above], then your screen shows the end of your buffer contents; when the mode line shows [More below], then your screen shows the beginning of your buffer contents. When it says [More above and below], then the buffer contents extend above and below the part that the screen displays. When the display shows the entire buffer contents, this message does not appear at all.

Mode Line Example

```
ZMACS (Text) text.text /dess/doc/books/ VAX: * [More above and below]
```

In this sample mode line, we are in Zmacs Text Mode, editing a file named text.text, which resides in the directory /dess/doc/books on the host named VAX. The file has been changed since we last saved it (indicated by the *), and the file contents extend above and below the portion that Zmacs displays on the screen.

Inserting Text

Introduction to Inserting Text

To insert new text anywhere in the buffer, position the cursor at the place you want the new text to go and type the new text. Zmacs always inserts characters at the cursor. The text to the right of the cursor is pushed along ahead of the text being inserted.

Inserting Characters

When you type in new text, you are actually issuing Zmacs commands. Ordinary printing characters are called *self-inserting* because when you type one, it inserts itself into the text in your buffer.

You can give numeric arguments to the keystrokes that insert printing characters into the buffer; Zmacs interprets these arguments as repeat counts. See the section "Numeric Arguments", page 26.

Example: `c-80 *` inserts a line of 80 asterisks at the cursor.

Starting a New Line

Newline characters delimit lines of text. They have no visible printed form, but are present at each line break. You can break one line into two lines by inserting a newline (pressing RETURN) where desired. Similarly, you can merge two lines into one by deleting the intervening newline.

Correcting Typos

To correct text you have just inserted, use the RUBOUT key. RUBOUT deletes the character *before* the cursor (not the one over which the cursor is positioned; that is the character *after* the cursor). The cursor and the rest of that line move to the left.

When given a numeric argument, RUBOUT saves the succession of deleted characters.

Example: `c-20 RUBOUT` kills the previous 20 characters and saves them together.

See the section "Deleting Vs. Killing Text", page 84.

When the cursor is positioned on the first character on a line and you press RUBOUT, the preceding newline character is deleted and Zmacs appends the text on that line to the end of the previous line.

Inserting Text, *cont'd.*

Wrapping Lines

When you add too many characters to one line without breaking it with a RETURN, the line grows to occupy two (or more) lines on the screen, with an exclamation point at the extreme right margin of all but the last of them. The ! means that the following screen line is not really a distinct line in the file, but just the continuation of a line too long to fit the screen.

Inserting Formatting Characters

You can insert most characters directly into the buffer by simply typing them, but other characters act as editing commands and do not insert themselves. If you need to insert a character that is normally a command (for example, TAB or RUBOUT), use the c-Q (Quoted Insert) command first to tell Zmacs to insert the following character into the buffer literally. c-Q prompts in the echo area for the character to be inserted and inserts it into the text.

Numeric Arguments

Overview of Numeric Arguments

Many Emacs commands take numeric arguments, which you type before the main command keystroke. Specify a numeric argument by pressing any combination of any of the modifier keys (c-, m-, s-, or h-) with the number. This way, you can type sequences of commands more easily without frequently alternating keys.

Numeric arguments to commands appear in the echo area when you do not type the command immediately. With no delay, the argument does not appear.

In general, use negative arguments to tell a command to move or act backwards. You can specify a negative argument by pressing any modifier key with the minus sign followed by the number. Most commands treat a numeric argument consisting of just a minus sign the same as -1.

Example of Numeric Arguments

c-F is the command to move the cursor forward one character.
 c-3 c-5 c-F moves point forward 35 characters;
 c-- c-3 c-5 c-F moves point backward 35 characters.

Throughout this manual, instead of writing out c-4 c-5 c-F or m-4 m-5 m-B, we usually abbreviate to c-45F or m-45B.

Defaults to Numeric Arguments

Many commands have default numeric arguments. This means that in the absence of a numeric argument, the command behaves as if the default argument were given. Most commands have a default argument of 1. This includes all the commands that interpret numeric arguments as repeat counts. Some commands have a different default and still others have no default: their behavior in the absence of a numeric argument is different from their behavior with a numeric argument.

c-U

Quadruple Numeric Arg

This special command prefixes other commands, usually representing a numeric argument of 4. You can repeat c-U; it multiplies the numeric argument by 4 each time. For example, c-U c-U c-F moves point forward 16 characters. Sometimes instead of representing a numeric argument of 4, c-U alters the action of a command slightly; for example, when used with the

Numeric Arguments, *cont'd.*

command Set Pop Mark, c-U takes different actions with the mark. (For a description of the Set Pop Mark command: See the section "Working with Regions in Zmacs", page 99.)

Introduction to Moving the Cursor

Description of Moving the Cursor

To do more than insert characters, you have to know how to move the cursor.

For complete descriptions of the commands summarized here and other cursor-moving commands: See the section "Moving the Cursor in Zmacs", page 63.

Summary of Moving the Cursor

c-A	Beginning of Line
Moves to the beginning of the line.	
c-E	End of Line
Moves to the end of the line.	
c-F	Forward
Moves forward one character.	
c-B	Backward
Moves backward one character.	
m-F	Forward Word
Moves forward one word.	
m-B	Backward Word
Moves backward one word.	
m-E	Forward Sentence
Moves to the end of the sentence in text mode.	
m-A	Backward Sentence
Moves to the beginning of the sentence in text mode.	
c-N	Down Real Line
Moves down one line.	
c-P	Up Real Line
Moves up one line.	
m-]	Forward Paragraph
Moves to the start of the next paragraph.	
m-[Backward Paragraph
Moves to the start of the current (or last) paragraph.	
c-X]	Next Page
Moves to the next page.	
c-X [Previous Page
Moves to the previous page.	

Introduction to Moving the Cursor, *cont'd.*

<code>c-V, SCROLL</code>	Next Screen
Moves down to display the next screenful of text.	
<code>m-V, m-SCROLL</code>	Previous Screen
Moves up to display the previous screenful of text.	
<code>m-<</code>	Goto Beginning
Moves to the beginning of the buffer.	
<code>m-></code>	Goto End
Moves to the end of the buffer.	

Introduction to Erasing Text

Description of Erasing Text

Most commands that erase text from the buffer save it so that you can get it back if you change your mind, or move or copy it to other parts of the buffer. These commands are known as *kill* commands. The rest of the commands that erase text do not save it; they are known as *delete* commands. The delete commands include `c-D` and `RUBOUT`, which delete only one character at a time, and those commands that delete only spaces or line separators. (However, when given a numeric argument, `c-D` and `RUBOUT` do save that sequence of deleted characters on the kill ring.) Commands that can destroy significant amounts of information generally kill. The commands' names and individual descriptions use the words "kill" and "delete" to say which they do.

If you issue a kill command by mistake, you can retrieve the text with `c-Y`, the Yank command. For details on killing and retrieving text: See the section "Working with Regions in Zmacs", page 99.

Summary of Erasing Text

<code>c-D</code>	Delete Forward
Deletes the character after point.	
<code>RUBOUT</code>	Rubout
Deletes the character before point.	
<code>m-D</code>	Kill Word
Kills forward one word.	
<code>m-RUBOUT</code>	Backward Kill Word
Kills backward one word.	
<code>m-K</code>	Kill Sentence
Kills forward one sentence.	
<code>c-X RUBOUT</code>	Backward Kill Sentence
Kills backward one sentence.	
<code>c-K</code>	Kill Line
Kills to the end of the line or kills an end of line.	
<code>c-W</code>	Kill Region
Kills region (from point to mark).	
<code>c-m-K</code>	Kill Sexp
Kills forward over exactly one Lisp expression.	

Introduction to Erasing Text, *cont'd.*

<code>c-m-RUBOUT</code>	Backward Kill Sexp
Kills backward over exactly one Lisp expression.	
<code>m-\</code>	Delete Horizontal Space
Deletes any spaces or tabs around point.	
<code>c-X c-0</code>	Delete Blank Lines
Deletes any blank lines following the end of the current line.	
<code>m-^</code>	Delete Indentation
Deletes RETURN and any indentation at front of line.	

Creating and Saving Buffers and Files

Description

You do all your text editing in Zmacs *buffers*, which are temporary workspaces that can hold text. To keep any text permanently you must put it in a *file*. Files store data for any length of time.

To edit the contents of a file using Zmacs, you create a buffer and copy the file contents into it. To add text to the end of the buffer, move point to the end of the buffer and type the new text. Editing proceeds in the buffer, not in the file. The file remains unchanged until you explicitly write the modified buffer contents to the file.

If you create multiple buffers, Zmacs keeps track of which files you are editing in which buffers. This association allows you to use completion to switch among buffers while you are editing them; you do not have to type the file name more than once. Zmacs always displays the name of the file you are currently editing.

The information in this section allows you to find or create and save a file. For complete information on buffers and files: See the section "Manipulating Buffers and Files in Zmacs", page 129.

Summary

c-X c-F	Find File
Reads the specified file into a buffer.	
c-X c-S	Save File
Saves out the changes to the current file.	
c-X B	Select Buffer
Selects the specified buffer.	
c-X c-W	Write File
Writes out the buffer to the specified file.	

Creating a Buffer

Zmacs creates your initial buffer when you first enter the editor. To create other buffers, use c-X B, Select Buffer, to create an empty buffer or c-X c-F, Find File, to create either an empty buffer or a buffer containing a file.

c-X B prompts for the name of the buffer to which you want to go. Type the buffer name and RETURN. If the buffer exists, Zmacs switches to that buffer and displays it on the screen. If the buffer does not already exist, Zmacs offers to let you create it by terminating the buffer name with c-RETURN. When you create a new (empty) buffer, the display is blank.

Creating and Saving Buffers and Files, *cont'd.*

The other way to create another buffer is `c-X c-F`, Find File. (`c-X c-F`) is described in detail in "Editing Existing Files". `c-X c-F` prompts for the name of a file, terminated by RETURN.

When you type `c-X c-F` for the first time in a Zmacs session, Zmacs offers you, as a default file name, an empty file (with the Lisp suffix native to your host computer) in your home directory on your host computer. For example:

<i>System</i>	<i>Empty Buffer Name</i>
Lisp Machine	foo.lisp
UNIX	foo.l
VMS	foo.lsp

Base and Syntax Default Settings for Lisp

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Zetalisp. See the section "Buffer and File Attributes in Zmacs", page 155.

Buffer Contents with `c-X c-F`

The first time you use `c-X c-F`, you can create an empty buffer using the Zmacs default file name, create an empty buffer using a name that you specify, or create a buffer containing an existing file:

- To create an empty buffer with the initial default file name as the one Zmacs associates with your buffer, press RETURN.
 - To create a new empty buffer, respond with any name. Zmacs switches to an empty buffer, gives the buffer the new name, and displays (New File) in the echo area.
 - To create a new buffer containing some file, respond to the prompt with the name of that file. Zmacs switches to an empty buffer, reads that file in, and names the buffer appropriately.
-

Creating and Saving Buffers and Files, *cont'd.*

Saving a File

Once you have the file in your buffer, you can make changes and then *save* the file with `c-x c-S`, the Save File command. This makes the changes permanent and actually changes the file. Until then, the changes are only inside your Zmacs buffer and the file itself is not really changed.

Creating a File

The first time you save or write the buffer, Zmacs creates the new file. You can create a new file with `c-x c-S`. Since a new file does not have a name associated with it yet, Zmacs asks for a name for the new file. It offers a *default pathname*, which is the name of the buffer. If you wish to save the file out to the default pathname, simply type a RETURN in response to the prompt.

If you wish to save the buffer in another file, provide that name as your response. Completion is offered to simplify your response.

You can also write the buffer out with `c-x c-W`, Write File. Zmacs prompts in the minibuffer for the name of the place you want to write the buffer's contents. `c-x c-W` also offers a default pathname, in this case, the name you supplied with `c-x c-F`.

Editing Existing Files

To tell Zmacs to edit text in a file, use `c-x c-F`, the Find File command, and give Zmacs a file name. You can enter the *pathname* of any file on any host that is reachable by network connections from your Lisp Machine. If the file already exists, Zmacs locates the file and reads it into your buffer.

Zmacs Commands for Formatting Text

Introduction

The extended commands `Format Region (m-X)`, `Format Buffer (m-X)`, and `Format File (m-X)` display text in a formatted style using formatting instructions that you embed in the text. You can send the formatted text to a supported printer (LGP1, LGP2, or DMP1) by giving the `Format` command a numeric argument. This prompts for an output device.

Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Zmacs also supports a number of commands for using character styles which allow text formatting of a somewhat different sort. See the section "Character Styles in Zmacs".

Producing Formatted Text

Producing formatted text requires two steps:

1. Entering the text and formatting instructions
2. Formatting that text with one of the Zmacs formatting commands

First you use the Zmacs editor to enter the text and embed formatting instructions, which can be *environments* and *commands*. These instructions format the text by, for example, specifying fonts, creating bulleted lists, and inserting headings.

For example, to specify that you want to italicize a group of words, like the title of a book, use the italicize environment. To emphasize a word, you might use the boldface environment.

This text:

```
@i(Gone With the Wind), by Margaret Mitchell, is a @b(great) book.
produces this, when formatted:
```

```
Gone With the Wind, by Margaret Mitchell, is a great book.
```

Formatting instructions all begin with an `@`. The "i" tells the formatter that you want the italicize environment, and the parentheses (*delimiters*) enclose the text within that environment. Other valid delimiters can be `()`, `[]`, `<>`, `{}`, `"`, `'`, or `”`.

How to Create an Environment

Environments can be either short form, `@i(italicize this)`, or long form, where the commands `@begin(i)` and `@end(i)` act as delimiters for the text that they enclose. For example, to italicize an entire passage:

Zmacs Commands for Formatting Text, *cont'd.*

`@begin(i)`

Environments can be either short form or long form. The long form uses the commands `@@begin` and `@@end` to act as delimiters for the text that they enclose.

`@end(i)`

produces this:

Environments can be either short form or long form. The long form uses the commands @begin and @end to act as delimiters for the text that they enclose.

(The @s inside the environment must be doubled so the formatter does not interpret them as format commands.)

The following environment *enumerates*, that is, numbers sequentially each separate line of text within it:

`@begin(enumerate)`

Paragraph 1

Paragraph 2

Paragraph 3

`@end(enumerate)`

produces the following output:

1. Paragraph 1
 2. Paragraph 2
 3. Paragraph 3
-

Basic Text

Formatting Environments

Environments can be either filled or unfilled:

Filled

Fills each output line to capacity within the limits of the display.

Unfilled

Keeps output lines exactly as you entered them, as in an example.

Basic formatting environments are:

b

Displays enclosed text in boldface.

c

Displays enclosed text in capital letters.

Zmacs Commands for Formatting Text, *cont'd.*

<code>center</code>	Centers each line in an unfilled environment.
<code>description</code>	Outdents paragraphs with single spacing and wider margins in a filled environment.
<code>display</code>	Displays enclosed text in Roman (default) typeface and widens both margins in an unfilled environment.
<code>enumerate</code>	Moves the left margin to the right, displaying a number in the left margin for each paragraph.
<code>equation</code>	Displays equations in an unfilled environment using fixed-width typeface. It widens both margins.
<code>example</code>	Displays examples in an unfilled environment using fixed-width typeface. It widens both margins.
<code>figure</code>	Displays figures in an unfilled environment using Roman (default) typeface with no changes to the margins.
<code>flushleft</code>	Displays unfilled text aligned at the left margin. It recognizes and includes leading spaces.
<code>flushright</code>	Displays unfilled text aligned at the right margin. It ignores trailing spaces.
<code>format</code>	Displays enclosed text in an unfilled environment using Roman (default) typeface with no changes to the margins. Any horizontal alignment that is needed should be done with tabbing commands (for example, <code>@\</code> and <code>@></code>).
<code>fullpagefigure</code>	Displays figures in an unfilled environment using Roman (default) typeface with no changes to the margins.
<code>fullpagetable</code>	Displays tables in an unfilled environment using Roman (default) typeface with no changes to the margins.
<code>g</code>	Displays enclosed text in Greek typeface.
<code>heading</code>	Centers each line in boldface type in an unfilled environment.

Zmacs Commands for Formatting Text, *cont'd.*

<code>i</code>	Displays enclosed text in italics.
<code>itemize</code>	Moves the left margin to the right, displaying a bullet in the left margin for each paragraph.
<code>majorheading</code>	Centers each line in boldface type in an unfilled environment.
<code>multiple</code>	Keeps enclosed text together as a single item within <code>itemize</code> , <code>enumerate</code> , or <code>description</code> environments, regardless of intervening paragraph breaks.
<code>outputexample</code>	Displays typeout examples in an unfilled environment using fixed-width typeface. It widens the right margin.
<code>p</code>	Displays enclosed text in bold italics.
<code>quotation</code>	Displays enclosed text in a filled environment using Roman (default) typeface. It widens both margins.
<code>r</code>	Displays enclosed text in Roman typeface. For example to override the default typeface of the <code>italicize</code> environment: <code>@I(The Iliad @r[and] The Odyssey)</code> produces <i>The Iliad</i> and <i>The Odyssey</i> .
<code>subheading</code>	Displays each line in boldface type in an unfilled environment flush to the left margin.
<code>table</code>	Displays tables in an unfilled environment using Roman (default) typeface with no changes to the margins.
<code>text</code>	Displays enclosed text in a filled environment.
<code>verbatim</code>	Displays enclosed text in an unfilled environment in fixed-width typeface with no changes to the margins.
<code>t</code>	Displays enclosed text in fixed-width typeface.

How to Use Formatting Commands

Formatting commands control the format of the text (such as blank spaces between lines, tab settings, line breaks) and whether the formatter centers the text or aligns it against one of the margins.

Zmacs Commands for Formatting Text, *cont'd.*

For example:

```
@i(Gone With the Wind),@* by Margaret Mitchell @# is a @b(great) book.
```

produces:

```
Gone With the Wind,  
by Margaret Mitchell is a great book.
```

The @* command forces a line break and the @# command leaves a blank em-space for a special character to be drawn in.

Some commands, like the @* in the example, are complete by themselves. Others accept arguments, which must be enclosed in delimiters. There is no such thing as a long form for a command; you cannot say @begin(blankspace) for example.

Basic Text Formatting Commands

@blankspace	On paper, leaves the specified amount of blank space on the page (for example, @blankspace(1line)). Distance can be specified with: in, inch, inches, " cm, centimeters mm, millimeters pt, pts, point, points pica, picas em, ems, quad, quads char, chars, character, characters, en, ens line, lines On the screen, display truncates the blank space to roughly one inch of vertical space.
@caption	Creates a figure caption enclosed in square brackets, for example, use @caption[This is the caption] to produce: [Figure Caption: This is the caption.]
@foot	Puts in a parenthetical note. Does not create bottom-of-the-page footnotes or numbering.
@note	Puts in a parenthetical note.
@tabclear	Clears all tabs. It takes no arguments: use @tabclear().

Zmacs Commands for Formatting Text, *cont'd.*

@tabdivide	Sets tabs to divide text into the specified number of columns, for example, <code>@tabdivide(4)</code> sets tabs to divide the following text into 4 columns across the page.
@tabset	Sets one or more tabs at specified positions. Distance can be specified with: in, inch, inches, " cm, centimeters mm, millimeters pt, pts, point, points pica, picas em, ems, quad, quads char, chars, character, characters, en, ens line, lines

These punctuation-character commands consist of an @ followed by one punctuation character. They take no arguments.

@#	Leaves a blank space (quad space or em-space) for a special character.
@*	Forces a line break.
@.	Generates a period and forces a single significant space after it (used for abbreviations).
@=	Sets a tab at the left side of text to be centered. Do not use in a filled environment. Works with the tab commands (<code>@\</code> , <code>@></code> , or <code>@=</code>).
@>	Sets a tab at the left side of text to be flushed right. Do not use in a filled environment. Works with the tab command (<code>@\</code> , <code>@></code> , or <code>@=</code>).
@\	Moves the cursor to the next tab stop or marks the end of text being centered or flushed right. Do not use in a filled environment.
@^	Sets a tab at the current cursor position. Do not use in a filled environment.
@@	Inserts an @ in the text.
@-	Ignores all the white space between it and the next text in the source.

Zmacs Commands for Formatting Text, *cont'd.*

Example of Using Tabs to Format Text

This example shows how to use tab stops to:

- Divide text into four columns
- Center text
- Flush right text
- Reset tabs

```
@begin(format)
@tabdivide(4)
1.@\*\@\*\*\*
2.@=a@\@=b@\@=c@\@=d
3.@=e@\@=f@\@=g@\@=h
4.Left@=Center@>right
5.Left@=Center@>right@
@tabclear()
6.Left@=Center@>right
@end(format)
```

produces:

```
1.          *           *           *
2.      a           b           c           d
3.      e           f           g           h
4.LefCenter
5.LefCenter           right
6.Left           Center           right
```

Zmacs Format Commands

The second (and final) step in formatting is to issue one of the formatting commands, which interprets the text and formatting instructions into the formatted text.

You can use the commands to format the text on your screen or on a printer. Check first on the screen before sending output to a printer.

Use a numeric argument to send the output to an LGP1, LGP2, or DMP1. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Zmacs Commands for Formatting Text, *cont'd.*

Format Region

Format Region (m-X)

Displays the contents of the region formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Format Buffer

Format Buffer (m-X)

Displays the contents of the buffer, formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Format File

Format File (m-X)

Displays the contents of the file, formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Executing CP Commands From Zmacs

If you wish to execute a CP command while editing, you can press SUSPEND and issue the command in the typeout window, you can press SELECT-L and issue the command to a Lisp Listener, or you can issue the `m-X Execute CP Command` command.

Execute CP Command

`m-X Execute Cp Command`

Reads a CP command line from the minibuffer and executes that command. All output from the command appears in the Zmacs typeout window.

Leaving Zmacs

Overview of Leaving Zmacs

Use a system-wide command to switch programs, such as SELECT, FUNCTION S, the System menu, or, if you have multiple windows on the screen, position the mouse to another window and click.

Leaving Zmacs with the SELECT Key

A set of windows is always available by pressing the SELECT key and then one of the following keys:

<i>Key</i>	<i>Program</i>
C	Converse, for messages to other users
D	Document Examiner, for examining documentation
E	Editor, the Zmacs text and program editor
F	File system editor for access to files and directories
I	Inspector, for inspecting and modifying data structures
L	Lisp
M	Mail reading and sending system
N	Notifications, for rereading system notifications
P	Peek, a system status display
T	Telnet, a virtual terminal utility for logging in to other hosts
X	Flavor Examiner, for examining the structure of flavors that are defined in the Lisp environment

Leaving Zmacs Via the System Menu

The System menu is a momentary menu that lists several choices for acting upon windows and calling programs (for example, a Lisp Listener, Zmacs, or the Inspector). You can always call the System menu by clicking [(R2)] (the right mouse button twice or holding down the SHIFT key and clicking right once). Use the System menu to do many things, among them:

- Create new windows.
 - Select old windows.
 - Change the size and placement of windows on the screen.
 - Hardcopy a file.
-

Leaving Zmacs, *cont'd.*

Leaving Zmacs with `c-z`

The Zmacs command `c-z` returns you to the window in which the `ed` function was most recently called, usually the Lisp Listener.

3. Getting Help in Zmacs

Getting Out of Trouble

Overview of

Getting Out of Trouble

Sometimes you type the wrong command. Mostly it is obvious what you have done wrong, and it is a simple matter to undo it. There are, however, some kinds of trouble you can get into that require special remedies. For example, you might accidentally delete large chunks of text you need or you might begin to type a command and then change your mind.

This section tells you how to recover from these situations.

Getting Out of Prefixes and Prompts

Most of the commands we have described are single keystrokes, but some keystrokes are prefixes that must be completed with a second keystroke to specify a command. `c-X` is the most important of these.

Getting Out of Keystroke Prefixes

If you press a `c-X` and don't mean it, you can get out by pressing either `c-G` or `ABORT`. These are general "get me out of here" commands, which you should use whenever you get yourself into a confused state. `ABORT` and `c-G` are, for the most part, synonymous in Zmacs.

Getting Out of Minibuffer Prompts

Sometimes you accidentally type a command that prompts for some additional information, or you type such a command on purpose and change your mind afterwards. When Zmacs prompts and you just want to get out of the minibuffer and back to where you were, press `ABORT`. If, instead, you wish to cancel and reenter your response, use `c-G`, which clears any typein but leaves you still in the minibuffer. When the minibuffer is empty, `c-G` cancels the minibuffer command. (With some echo area prompts, you have to use `ABORT`.)

`ABORT`

Abort At Top Level

Cancels the last command typed. It also cancels numeric arguments and region marking.

Getting Out of Trouble, *cont'd.*

c-G Beep
 Cancels the last command. It also cancels numeric arguments and region marking, except when given an argument. It cancels one thing at a time, so that if you've typed a number of commands or responses, you must use successive c-Gs to cancel each one and return to top level.

Large Deletions

Do not delete large pieces of text by repeatedly pressing RUBOUT and c-D. Apart from being slow, text deleted character-by-character is gone for good.

Instead, use delete and kill commands that save deleted regions in the kill history. c-K, m-K, and the commands that deal with *regions* easily wipe out and save larger chunks. Also, RUBOUT or c-D with a numeric argument erases that many characters all at once and saves them in the kill history. For full descriptions of these delete and kill commands: See the section "Deleting and Transposing Text in Zmacs", page 83.

Getting Text Back

The system has different histories for different contexts. One of these is always the *current history*. The two histories that you need to use for yanking in Zmacs are the *kill history* and the *command history*. The kill history remembers pieces of text that you killed or copied into it. In the context of Zmacs, the command history remembers all the editor commands that use the minibuffer in any way.

Additions to the histories are placed at the top of the list, so that history elements are stored in reverse chronological order – the newer elements at the top of the history, the older elements toward the bottom. A history remembers everything that has been typed to it since the last cold boot – it has no size limit.

Yanking commands pull in the elements of the history. *Top-level commands* start a yanking sequence; for example, c-Y yanks back the last text killed from the kill history, and c-m-Y yanks back the last command performed in the minibuffer. m-Y performs all subsequent yanks in the same sequence; for example, pressing m-Y while the kill history is the current history yanks the next item from that history.

A yanking sequence ends when you type new text, execute a form or command, or start another yanking sequence.

For complete descriptions of killing and yanking: See the section "Working with Regions in Zmacs", page 99.

Finding Out About Zmacs Commands

Overview of Finding Out About Zmacs Commands

Sometimes you want to know if a Zmacs command exists that performs a certain function. Or, you might think that you know what a certain keystroke does, but you still want to make sure, or refresh your memory about its exact usage. This manual is one resource you might use in these circumstances. Zmacs itself has a number of built-in self-documentation facilities. This section describes some ways to get at this documentation.

Finding Out About Zmacs Commands with HELP

The HELP key is a prefix to a useful group of commands giving various kinds of online help. If you forget what a command does, which keystrokes perform an action, or have no idea how to accomplish something, press HELP.

Whenever you have a question of any kind, press HELP. Zmacs prompts you in the minibuffer for details on what kind of help. If you don't know, press HELP again and it tells you, in the *typeout window*, how to find what you're looking for. The typeout window displays right over the editor window. The actual contents of the buffer are not affected, and the next command you type restores the buffer display.

Finding Out What a Zmacs Command Does

HELP C

The command HELP C displays "Document Command:" below the mode line and waits for you to type a command. When you do, Zmacs displays the internal documentation for that command.

Example

If you press HELP-C followed by c-F, the response is:

```
c-F is Forward, implemented by COM-FORWARD:
Moves forward one character.
With a numeric argument (n), it moves forward n characters.
```

The first line above tells you the name of the command (in this case Forward), and the name of the internal Lisp function that actually does the work (in this case **zl-user:com-forward**). (You

Finding Out About Zmacs Commands, *cont'd.*

don't need to know these internal names for basic editing.) The COM-xxx name displayed by HELP C is mouse-sensitive: clicking left on it edits the COM-xxx function, and clicking right displays a menu with choices of Arglist, Edit, Disassemble, and Documentation.

The next line is a very short description of what the command does; it usually tells you what the command does without a numeric argument and how a numeric argument modifies that behavior.

Finding Out What a Prefix Command Does

When you ask (with HELP C) for documentation on a prefix command like c-X, Zmacs prompts you, in the typeout window, to complete the command. Zmacs displays the documentation for the prefix command in the typeout window.

Finding Out What an Extended Command Does

HELP D

When you want to find out what an extended command does, you can display the documentation for the command by pressing HELP D, which prompts in the minibuffer "Describe command:", to which you type the command's name.

Searching for Appropriate Zmacs Commands

HELP A

m-X Apropos

When you can only guess at part of the name or function of a command by the action it performs, there is a command, HELP A, to help you scan information about all the available Zmacs commands to find the one you want. All you have to do is type in a string, such as "buffer" and all command names plus the first line of all help documentation are scanned for the string you specify.

Each Zmacs command has a name. The name is almost always exactly what you would expect; that is, the name describes the function of the command in reasonably plain English. If not, the word you're looking for is almost surely in the first line of the help documentation.

Finding Out About Zmacs Commands, *cont'd.*

With a numeric argument, HELP A searches only the command names.

The A stands for apropos. The $m-X$ Apropos command works the same way.

Method for Searching for Appropriate Zmacs Commands

To find the command you want, just press HELP A or type $m-X$ Apropos. Zmacs prompts you for a substring, you enter your guess, and then Zmacs displays short descriptions of all the commands whose names contain that substring. If the substring that you enter contains a space, then Zmacs displays a short description of all the commands whose names or help documentation includes a similarly positioned space. Each description gives the short documentation for the command and tells what keystrokes invoke it.

Example of a Search String for HELP A

The command you perform when you use $m-Q$ is called "Fill Paragraph", so you might expect a command that counts the number of paragraphs in the buffer to be called something like "Count Paragraphs" or "Paragraphs Count". No matter what, the word *paragraph* is going to be in the name or the first line of the help documentation.

Finding Out What You Have Typed

HELP L

As you are editing you might find yourself in a hopelessly confused state and not know how to recover.

If this happens to you it is often very enlightening to press HELP L to list the last 60 keystrokes you typed. By examining your own recent activity, it is often possible to find out where you went wrong and how to save yourself.

More HELP Commands for Finding Out About Zmacs Commands

Finding Out About Zmacs Commands, *cont'd.*

HELP U

Offers to undo the last "major" operation (such as fill or sort).

HELP V

Displays all the Zmacs variables whose names contain a certain substring. For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.

HELP W

Finds out whether an extended command is bound to a key.

General Information-giving Zmacs Commands

The following commands display:

- Information about the location of point
- Documentation about a specified Lisp function
- Argument list for the specified Lisp function
- Information about the current Lisp variable
- The number of lines in the region or page
- Possible parenthesis mismatches
- Trace information about the specified Lisp function

The word *current*, when describing a Lisp function or a Lisp variable, refers to (approximately) the function or variable whose name is nearest to the cursor.

c-X =

Where Am I

Displays various things about the location of point. It displays the X and Y positions, the octal code for the following character, the current line number and its percentage of the total file size. If there is a region, it displays the number of lines in it. Fast Where Am I (c-=) displays a subset of this information more quickly.

c-=

Fast Where Am I

Quickly displays various things about where point is. It displays the X and Y positions and the octal code for the following character. If there is a region, it displays the number of lines in it. Where Am I displays the same things and more.

Finding Out About Zmacs Commands, *cont'd.*

`m-sh-D` Show Documentation

Displays the documentation for the given topic. It prompts for a topic name offering completion only on topics in the documentation database. With a numeric argument, `m-sh-D` directs the display to either the screen or paper (hardcopy).

See the section "The Document Examiner" in *User's Guide to Symbolics Computers*.

`c-sh-D` Long Documentation

Displays the documentation string for the specified function. It prompts for a function name, which you can either type in or select with the mouse. The default is the current function.

When this command does not find a documentation string, it suggests you use Show Documentation (`m-X`) or the Document Examiner to see the function's online documentation.

`c-sh-A` Quick Arglist

Displays the argument list for the current function. With a numeric argument, it reads the function name from the minibuffer.

Arglist (`m-X`)

Displays the argument list of the specified function. It reads the name of the function (from the minibuffer) and displays the argument list in the echo area.

`c-sh-V` Describe Variable At Point

Displays information in the echo area about the current Lisp variable. The information displayed shows whether it is declared special, whether it has a value, and whether it has documentation put on by `defvar`. When nothing is available, it checks for lookalike symbols in other packages.

`m==` Count Lines Region

Displays the number of lines in the region.

`c-X L` Count Lines Page

Displays the number of lines on the current page (or the buffer, if there are no page delimiters). In parentheses, it displays the number of lines up to the line containing point and the number of lines after the line containing point.

Finding Out About Zmacs Commands, *cont'd.*

Find Unbalanced Parentheses (m-X)

Finds any parenthesis mismatch error in the buffer. It reads through all of the current buffer and tries to find places in which the parentheses do not balance. It positions point to possible trouble spots, printing out a message that says what the trouble appears to be. This command finds only one such error; if you suspect more errors, run it again.

Trace (m-X)

Traces or untraces a function. It reads the name of the function from the minibuffer and then it pops up a menu of trace options. With an argument, it omits the menu step.

See the special form **trace** in *Program Development Utilities*.

See the section "Options to **trace**" in *Program Development Utilities*.

The Editor Menu

Overview of the Editor Menu

Click right in Zmacs to display the *editor menu*, a momentary menu containing editor commands, each of which is a possible choice. Position the mouse cursor over an item and then click the appropriate button to make the choice.

For complete descriptions of the editor menu commands: See the section "Editor Menu Commands", page 57.

Editor Menu Commands

The Editor Menu commands are:

<i>Command</i>	<i>Description</i>
Arglist	Prints the argument list of the specified function: See the section "General Information-giving Zmacs Commands", page 54.
Edit Definition	Prepares to edit the definition of a specified function: See the section "Editing Lisp Programs in Zmacs", page 223.
List Callers	Lists all functions that call the specified function: See the section "Editing Lisp Programs in Zmacs", page 223.
List Definitions	Displays the definitions in a specified buffer: See the section "Editing Lisp Programs in Zmacs", page 223.
List Buffers	Prints a list of all the buffers and their associated files: See the section "Manipulating Buffers and Files in Zmacs", page 129.
Kill Or Save Buffers	Offers a menu of modified files with choices to kill, save, or remove the modification flag from the file: See the section "Manipulating Buffers and Files in Zmacs", page 129.

The Editor Menu, *cont'd.*

Split Screen	Makes several windows split among the buffers as specified: See the section "Manipulating Buffers and Files in Zmacs", page 129.
Compile Region	Compiles the region, or if no region is defined, the current definition: See the section "Editing Lisp Programs in Zmacs", page 223.
Indent Region	Indents each line in the region: See the section "Changing Case and Indentation in Zmacs", page 213.
Change Default Font	Sets the default font: See the section "Working with Regions in Zmacs", page 99.
Change Font Region	Changes the font for the region: See the section "Working with Regions in Zmacs", page 99.
Uppercase Region	Changes any lowercase characters in the region to uppercase: See the section "Working with Regions in Zmacs", page 99.
Lowercase Region	Changes any uppercase characters in the region to lowercase: See the section "Working with Regions in Zmacs", page 99.
Indent Rigidly	Shifts text in the region sideways as a unit: See the section "Changing Case and Indentation in Zmacs", page 213.
Indent Under	Fixes indentation to align under either a character that you click on with the mouse cursor or a string read from the minibuffer: See the section "Aligning Indentation in Zmacs", page 220.

More on the Minibuffer

Minibuffer

Response Format

Most commands expect only one line of response. In these cases, the END key has the same meaning as the RETURN key, terminating the response.

However, for commands that expect one or more lines of response, RETURN has its usual significance, inserting a newline in the minibuffer, and END marks the end of the response.

Minibuffer

Response Help

While responding to a prompt, you can press HELP to get documentation describing the current situation. Zmacs tells you exactly what input it expects and what the possible responses are.

More Ways to

Enter Minibuffer Responses

Yanking and mousing provide quick and simple ways to enter minibuffer responses without having to type them out. Both of these methods are context-sensitive. Yanking works only when you have previously entered a minibuffer response. Mousing works when you click on a name that makes sense in the context of the minibuffer prompt.

Yanking in the Minibuffer

`c-n-Y`

Repeat Last Minibuffer Command

Repeats a recent minibuffer command. It yanks the displayed default if there is one, otherwise, it yanks the last thing typed in this context. A numeric argument *n* yanks the *n*th previous one. An argument of 0 lists the history of elements typed in the minibuffer.

For a similar command with string-matching: See the section "Retrieving History Elements", page 87.

After `c-n-Y`, `m-Y` replaces what was yanked with a previous element of the same history, in this case, another minibuffer command. For more details: See the section "Retrieving History Elements", page 87.

`m-Y`

Yank Pop

Corrects a yank to use a different element of its history. The

More on the Minibuffer, *cont'd.*

most recent command must be a yanking command (c-Y, m-Y, c-sh-Y, m-sh-Y or c-m-Y). The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive m-Y) to bring this element to the top.

A numeric argument of zero displays the history. A positive numeric argument of *n* moves *n* elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

Getting Information About Buffers and Regions

A good deal of information is available about each Zmacs buffer or region. You can get a count of characters, words, lines, paragraphs, or pages, as well as a count of substrings or Lisp objects.

Count Chars

`m-X Count Chars`

Counts the characters in the region or in the buffer if there is no region.

Count Words

`m-X Count Words`

Counts the words in the region or in the buffer if there is no region.

Count Lines

`m-X Count Lines`

Counts the lines in the region or in the buffer if there is no region.

Count Paragraphs

`m-X Count Paragraphs`

Counts the paragraphs in the region or in the buffer if there is no region.

Count Pages

`m-X Count Pages`

Counts the pages in the region, or in the buffer if there is no region.

Getting Information About Buffers and Regions, *cont'd.*

Count Occurrences

`m-X` Count Occurrences

Counts how many times a certain substring occurs in the region or in the buffer following point if there is no region.

See the section "How Many", page 62.

How Many

`m-X` How Many

Counts how many times a certain substring occurs in the buffer following point. You are prompted for the substring.

See the section "Count Occurrences", page 62.

4. Moving the Cursor in Zmacs

Overview of Moving the Cursor

Summary of Cursor Movement

To make changes at a particular place in a Zmacs buffer, you must move the cursor to that place, since most commands that modify the buffer do so immediately around the cursor.

The cursor movement or *motion* commands:

- View the contents of the buffer
 - Redisplay the editor window
 - Move the cursor around the buffer using mouse commands
 - Move the cursor around the buffer using keystroke commands
-

The Editor Window and the Buffer

The *editor window* displays either a portion of your buffer or the whole buffer, depending on the size of the buffer and your current location in it.

When the current buffer is smaller than the exact size of the editor window, Zmacs displays the contents of the buffer at the top of the window and leaves the bottom of the window blank. You cannot tell whether the buffer actually comes to an end where the text stops, since there could be white space and newline characters after the last visible piece of text.

When the buffer is too large to fit on the screen, the editor window shows only a section of the buffer. The part that shows always contains the cursor, so it never vanishes off the top or bottom of the editor window. Zmacs changes the position of the editor window inside the buffer as seldom as possible – usually only when you try to move the cursor off the top or bottom of the screen.

Wraparound Lines in the Editor Window

Lines that are too long to fit across the editor window are displayed on as many physical lines as are necessary. An exclamation point (!) in the (normally blank) last column means that the next physical line is part of the same logical line.

Redisplaying the Window

Introduction to Redisplaying the Window

Whenever you modify the buffer's contents or move point or the mark, Zmacs updates the display to reflect the change. (For a discussion of the mark: See the section "Working with Regions in Zmacs", page 99.) This updating can be as simple as moving the cursor or as involved as figuring out the whole display from scratch. These operations are called *redisplay* and Zmacs performs them automatically.

For example, when you move the cursor off the top or bottom of the editor window, a complete redisplay is required. The window has to shift to show a different part of the buffer in order to keep the cursor visible.

You can explicitly tell Zmacs to do a redisplay with the Recenter Window command, invoked by `c-L`. You might want to do this if the cursor gets too close to the top or the bottom of the editor window, and you want to redisplay with the cursor closer to the center so that you can see more context in one direction or the other.

It is important to remember that redisplay operations change only the *display*, not the actual contents of the buffer.

Recentering the Window

`c-L`

Recenter Window

Completely redisplay the screen, leaving the cursor near the middle of the editor window.

With a numeric argument of n , it leaves the cursor n lines from the top of the window. With a negative numeric argument of $-n$, it leaves the cursor n lines from the bottom of the window.

Displaying the Next Screen

`c-V`, SCROLL

Next Screen

Moves the cursor to the beginning of the last visible line in the editor window and redisplay the screen with that line at the top of the window.

With a numeric argument of n , it moves the text up n lines. With a negative numeric argument $-n$, it moves the text down n lines. The cursor does not move (with respect to the text) unless the numeric argument is large enough to slide it off the screen. In that case the cursor remains at the top.

Redisplaying the Window, *cont'd.*

Displaying the Previous Screen

`m-U`, `m-SCROLL`

Previous Screen

Moves the cursor to the beginning of the first visible line in the editor window and redisplays the screen with that line at the bottom of the window.

With a numeric argument of n , it moves the text down n lines. With a negative numeric argument $-n$, it moves the text up n lines. The cursor does not move (with respect to the text) unless the numeric argument is large enough to slide it off the screen. In that case the cursor remains at the bottom.

Positioning the Window Around a Definition

`c-m-R`

Reposition Window

Redisplays, trying to get all of the current function definition in the window. It puts the beginning of the current definition at the top of the window with the current position of the cursor still visible. Doing `c-m-R` twice pushes comments off the top of the window, making more of the code of a large function visible.

Moving to a Specified Line

`m-R`

Move To Screen Edge

Moves to the beginning of a specified line on the screen. With no argument, it moves to the beginning of a line near the middle of the screen. The exact line is controlled by the Zmacs variable Center Fraction. A numeric argument specifies a particular line to move to. Negative arguments count up from the bottom of the window. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

Moving the Cursor with the Mouse

Introduction to Using the Mouse

The easiest way to get the cursor where you want it is with the *mouse*. See the section "The Mouse" in *User's Guide to Symbolics Computers*.

Mouse Documentation Line in Zmacs

The mouse documentation line at the bottom of the screen tells you what will happen when you click the mouse. The small arrow cursor tells you where the mouse is and what it is over. What you can do with the mouse depends on what the mouse is over.

There are five sets of possible mouse actions, corresponding to four things the mouse can be over in a Zmacs buffer:

Text or blank space

Scroll bar

Upper left corner of screen

Upper right corner of screen

The mouse documentation line is two lines high. The top line tells you what you can do with the three mouse keys. The bottom line tells you what keyboard keys you can press to change the action of the mouse keys. When you press one of these keyboard keys, the top mouse documentation line tells you what you can do with the three mouse keys while that keyboard key is held down.

The three mouse keys are called **L** for left, **M** for middle and **R** for right.

The keyboard keys are called **c** for CONTROL, **Sh** for SHIFT, **s** for SUPER and **m-sh** for META-SHIFT.

CONTROL and SHIFT affect editing operations. Their actions are described in this section.

META-SHIFT with the right mouse button gives you access to the *window operation menu*, which allows you to move, reshape, expand, bury, kill, or hardcopy the window or pane.

SUPER with the right mouse button gives you access to the

Moving the Cursor with the Mouse, *cont'd.*

presentation debugging menu.

Mouse Over Text or Blank Space

Here is a description of what you can do with the mouse when it is over text or blank space in a Zmacs buffer.

<i>Notation</i>	<i>Description</i>
L:Move point	<p>Performs two separate actions, depending on whether you click left or hold left down.</p> <ul style="list-style-type: none"> • Relocates the cursor: position the mouse cursor to the desired location and click left. If the cursor is over blank space, point is moved to the end of the line. • Marks a region: position mouse cursor to desired location, hold left button down, move mouse cursor to end point of desired region and release the button .
sh-L:Move to point	Relocates the mouse arrow cursor to point (where the blinking cursor appears).
M:Mark thing	<p>Marks (makes into a region) the object on which you click. Clicking after the end of a line or before the first nonblank character of a line marks the whole line. Clicking on a word marks that word.</p> <p>In Lisp mode, however, if that word is part of what could be a symbol's printname, it marks that whole symbol name. Clicking on an open or close parenthesis marks all the text between that parenthesis and its matching parenthesis, including the parentheses. Clicking on an open or close quotation mark (") marks the whole quoted string. Clicking between words marks all text up to the end of the next word or possible symbol printname, depending on mode. For a complete description of <i>marking</i> regions: See the section "Working with Regions in Zmacs", page 99.</p>

Moving the Cursor with the Mouse, cont'd.

c-M:Copy Mouse

Inserts the object on which you click, as though you had typed it. This allows you to build a program or document by selecting things already appearing on your screen, in the manner of a menu. Hold down the control key and click middle on the object you want to copy: it is inserted as though you had just typed it. If you change your mind, and want to remove what you have just inserted, type c-W, and it is removed.

The object copied can be a word, a printed representation of a Lisp symbol, a parenthesized or quoted group of words, a printed representation of a lisp list or string, or a line. What object is picked up by clicking c-(M) on it is determined by the same rules as Mark Thing (M) in Lisp Mode. That is:

- Clicking after the end of a line or before the first nonblank character of a line copies the whole line. Clicking on a word picks up that whole word, or possible Lisp Symbol printname of which that word could be part.
- Clicking on an open or close parenthesis copies the text between that parenthesis and its matching parenthesis, including the parentheses. Clicking on an open or close quotation mark (") copies the whole quoted string. Clicking between words copies all text up to the end of the next word (or possible symbol printname).

Appropriate spaces are placed before the inserted object.

sh-M:Save/Kill/Yank Performs one of four related actions:

- If there is a region, it saves the region in the kill history while leaving it in the buffer (like m-W)
- If the last command saved the region, it

Moving the Cursor with the Mouse, *cont'd.*

wipes it from the buffer (like `c-W` except it does not save)

- If the above two conditions do not apply, it yanks the first element from the kill history (like `c-Y`)
- If the last command was a yank command, it yanks the next item from the kill history (like `m-Y`)

For a complete description of *saving*, *killing*, and *yanking* regions: See the section "Working with Regions in Zmacs", page 99.

- `R`:Editor menu Displays a Zmacs menu offering mouse-sensitive Zmacs commands.
- `sh-R`:System Menu Displays a System menu.

Mouse Over Scroll Bar

The scroll bar consists of a pair of dotted lines with a shaded section between them. The shaded section, or elevator, represents the portion of the buffer visible on the screen and the dotted lines, or shaft, represent the entire buffer. In general, you see only the shaft, but not the elevator. Move the mouse cursor over the shaft and the elevator appears. The elevator remains visible until the buffer changes in size, whereupon it disappears until you move the mouse cursor over the shaft again.

Motion Commands

Introduction to the Motion Commands

Zmacs word, sentence, and paragraph motion commands all have strict definitions for where words, sentences, and paragraphs begin and end. You can modify all these definitions.

Numeric Arguments and the Motion Commands

All of the motion commands allow numeric arguments. For the most part, these numeric arguments are interpreted as repeat counts.

Example of Numeric Arguments with Motion Commands

n -F moves the cursor forward one word, whereas n -13F moves the cursor forward 13 words.

Negative Numeric Arguments and Motion Commands

Most of the motion commands come in pairs, with one command for forward motion over a particular unit and one command for backward motion. Both kinds of commands often interpret negative numeric arguments by reversing the direction of motion.

These conventions – that Zmacs interprets numeric arguments as repeat counts, and that negative numeric arguments reverse the direction of motion – together make up the *motion convention*.

Example of Negative Numeric Arguments with Motion Commands

n - -13F moves point backward 13 words. n -13B has exactly the same effect.

Motion by Character

A Zmacs *character* can be any letter, number, or punctuation character.

Motion Commands, *cont'd.*

Forward Character

c-F

Forward

Moves the cursor forward over one character. c-F interprets numeric arguments as repeat counts.

Negative numeric arguments reverse the direction of motion. For example, c-3B and c- -3F both move the cursor backwards three characters.

Backward Character

c-B

Backward

Moves the cursor backward over one character. c-B interprets numeric arguments as repeat counts.

Negative numeric arguments reverse the direction of motion. For example, c-3 c-B and c-- c-3 c-F both move the cursor backwards three characters.

Motion by Word

Zmacs generally considers a *word* to consist of a sequential string of alphanumeric characters, that is, any combination of the characters a-z, A-Z, and 0-9. Different major modes define their own delimiter characters. For example, in Text Mode an apostrophe (') is part of a word, but in other modes it is a delimiter. (For mode descriptions: See the section "Setting the Zmacs Major Mode", page 175.)

Forward Word

m-F

Forward Word

Moves the cursor forward one word. Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-F always places the cursor at the end of a word. If the cursor is in the middle of a word, m-F moves the cursor to the end of that word.

Backward Word

m-B

Backward Word

Moves the cursor backward one word. Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

Motion Commands, *cont'd.*

m-B always places the cursor at the beginning of a word. If the cursor is in the middle of a word, m-B moves the cursor to the beginning of that word.

Motion by Sentence

Description of Zmacs Sentence Delimiters

According to Zmacs, sentences can end with question marks, periods, and exclamation points. Furthermore, these punctuation marks only end a sentence when followed by:

- A newline
- A space followed by either a newline or another space.

However, Zmacs allows any number of *closing characters*, which are ", ',), and], between the sentence-ending punctuation and the white space that follows it. A sentence also starts after a blank line.

This corresponds closely to standard typing conventions. Zmacs does not recognize a period followed by one space as the end of a sentence, for example, as in "e.g. " or "Dr. ".

Forward Sentence

m-E

Forward Sentence

Moves the cursor forward one sentence.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-E always places the cursor at the end of a sentence. If the cursor is in the middle of a sentence, m-E moves the cursor to the end of that sentence.

Backward Sentence

m-B

Backward Sentence

Moves the cursor backward one sentence.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-B always places the cursor at the beginning of a sentence. If the cursor is in the middle of a sentence, m-B moves the cursor to the beginning of that sentence.

Motion Commands, *cont'd.*

Motion by Line

Lines are delimited by special characters called *newlines*.

Down Line

c-N Down Real Line

Moves the cursor straight down to the corresponding column of the next line. If the cursor is positioned in the middle of the line, c-N moves it to the middle of the next one.

With a numeric argument n , it moves the cursor down n lines. Moving down a negative number of lines is the same as moving up.

Up Line

c-P Up Real Line

Moves the cursor straight up to the corresponding column of the previous line. If the cursor is positioned in the middle of the line, c-P moves it to the middle of the previous one.

With a numeric argument of n , it moves the cursor up n lines. Moving up a negative number of lines is the same as moving down.

Beginning of Line

c-A Beginning of Line

Moves the cursor to the beginning of the current line.

With a numeric argument of n , it moves the cursor to the beginning of the n th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

End of Line

c-E End Of Line

Moves the cursor to the end of the current line.

With a numeric argument of n , it moves the cursor to the end of the n th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

Goal Column and the Motion Commands

Motion Commands, *cont'd.*

Set Goal Column

c-X c-N

Set Goal Column

Sets the default column position (*goal column*). The goal column sets point position for c-N and c-P. It disables the default action of matching the goal column to point's current column and sets the goal column to zero instead. With a numeric argument *n*, sets the goal column to *n*. c-U turns it off (sets it back to the default state of keeping cursor in same horizontal position for c-N and c-P).

Motion by Lisp Expression

Description

Motion by Lisp expression repositions the cursor according to Lisp code delimiters: *lists* and *expressions*. A list is something enclosed in balanced parentheses. A Lisp expression is any readable printed representation of a Lisp object.

c-m-N Forward List

Moves forward over one list. It accepts a numeric argument for repetition count.

c-m-P Backward List

Moves backward over one list. It accepts a numeric argument for repetition count.

Motion Along One Nesting Level

Point always sits either between two expressions or in the middle of a Lisp object (excluding a list or nil).

c-m-F Forward Sexp

Moves point to the end of a surrounding Lisp object (excluding a list or nil) if there is one, or past the Lisp expression immediately to the right if not.

If parentheses are unbalanced to such an extent that it doesn't make sense to talk about "the expression on the right", this command gives an error message and does not move point at all.

c-m-F observes the motion convention for numeric arguments.

c-m-B Backward Sexp

Moves point to the beginning of a surrounding Lisp object (excluding a list or nil) if there is one, or to the beginning of the Lisp expression immediately to the left if not.

If parentheses are unbalanced to such an extent that it doesn't make sense to talk about "the expression on the left", this command gives an error message and does not move point at all.

c-m-B observes the motion convention for numeric arguments.

Motion by Lisp Expression, *cont'd.*

Motion up and Down Nesting Levels

c-m-D

Down List

Moves point forward past any intervening Lisp object (excluding a list or `nil`) to the level of list structure and leaves point just to the right of the open parenthesis of that expression.

With a numeric argument of n , it moves down n nesting levels.

c-m-U

Backward Up List

c-m-(

Backs up out of nesting levels. It moves backward one level of list structure. It searches for an open parenthesis and leaves point to the left of that open parenthesis. Also, if called inside of a string, it moves back up out of that string, leaving point to the left of its starting quote. It accepts numeric arguments for repetition count.

With a numeric argument of n , it moves up n nesting levels.

c-m-)

Forward Up List

Moves forward out of nesting levels. It moves forward one level of list structure. It searches for a close parenthesis and leaves point to the right of that close parenthesis. Also, if called inside of a string, it moves up out of that string, leaving point to the right of its ending quote. It accepts numeric arguments for repetition count.

With a numeric argument of n , it moves up n nesting levels.

Motion Among Top-Level Expressions

A Lisp file contains a sequence of expressions that we call *top-level expressions*, to distinguish them from their own subexpressions. Zmacs assumes that top-level expressions begin with an open parenthesis against the left margin. It does *not* parse top-level expressions by balancing parentheses, since parentheses do not always balance while programs are being written. The indentation represents the *programmer's* conception of program structure, and provides a better guide. So by *top-level expression*, we mean a section of text delimited by open parentheses at the beginning of two lines.

In code that includes a string containing a carriage return followed by an open parenthesis, show that the open parenthesis

Motion by Lisp Expression, *cont'd.*

does not start a top-level expression by putting a slash in front of it.

<code>c-n-A</code>	Beginning Of Definition
<code>c-n-[</code>	

Moves point to the beginning of the current top-level expression.

With a positive numeric argument n , it moves back n top-level expressions. With a negative numeric argument $-n$, it moves forward n top-level expressions.

<code>c-n-E</code>	End Of Definition
<code>c-n-]</code>	

Moves point to the end of the current top-level expression.

With a positive numeric argument n , it moves forward n top-level expressions. With a negative numeric argument $-n$, it moves back n top-level expressions.

<code>m-)</code>	Move Over)
------------------	-------------

Moves past the next close parenthesis, then does Indent New Line. It removes any whitespace between point and the close parenthesis before moving over it. With a positive argument n , after finding the next close parenthesis and deleting whitespace before it, it moves past $n-1$ additional close parentheses before doing Indent New Line. It ignores numeric arguments that are less than 1.

Motion by Paragraph

Introduction

A paragraph is delimited by:

- A newline followed by blanks (spaces or tabs)
 - A blank line
 - A Page character alone on a line
 - Various other mode-dependent factors (for example, a line that does not begin with the fill-prefix). See the section "Filling a Region", page 109.
-

Forward Paragraph

`m-]`

Forward Paragraph

Moves the cursor forward one paragraph.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

`m-]` always places the cursor at the end of a paragraph. If the cursor is in the middle of a paragraph, `m-]` moves the cursor to the end of that paragraph.

Backward Paragraph

`m-[`

Backward Paragraph

Moves the cursor one paragraph backward.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

`m-[` always places the cursor at the beginning of a paragraph. If the cursor is in the middle of a paragraph, `m-[` moves the cursor to the beginning of that paragraph.

Motion by Page

Introduction

Pages are delimited by Page characters. You can insert a Page character by pressing the PAGE key. The Page delimiter belongs to the page that precedes it and is therefore the last character on that page.

Forward Page

c-X]

Next Page

Moves the cursor to the beginning of the next page; that is, puts the cursor immediately after the nearest following Page delimiter. If the buffer does not contain a Page delimiter, it goes to the end of the buffer.

With a positive numeric argument n , it repeats this operation n times to move forward n pages. A negative numeric argument $-n$ moves the cursor backward instead.

c-X [always places the cursor immediately to the right of the next Page delimiter. If the cursor is immediately to the left of the Page delimiter, c-X] goes to the beginning of the page after next rather than just moving forward one character.

Backward Page

c-X [

Previous Page

Moves the cursor to the beginning of the previous page; that is, puts the cursor immediately after the nearest preceding Page delimiter. If the buffer does not contain a Page delimiter, it goes to the beginning of the buffer.

With a positive numeric argument n , it repeats this operation n times to move backward n pages. A negative numeric argument $-n$ moves the cursor forward instead.

c-X [always places the cursor at the beginning of a page. If the cursor is already at the beginning of the page, c-X [moves it to the beginning of the previous page.

Motion with Respect to the Whole Buffer

Beginning/End of Buffer

`m-<`

Goto Beginning

Moves the cursor to the beginning of the buffer.

With a numeric argument n between 0 and 10, it moves the cursor to a place $n/10$ of the way (counted in lines) from the beginning of the buffer towards the end.

`m->`

Goto End

Moves the cursor to the end of the buffer. You can use `m->` if you are in doubt as to the exact place on the screen where the buffer stops.

With a numeric argument n between 0 and 10, it moves the cursor to a place $n/10$ of the way (counted in lines) from the end of the buffer towards the beginning.

5. Deleting and Transposing Text in Zmacs

Deleting Vs. Killing Text

Overview

Deleting text merely gets rid of it, but Zmacs deletion commands not only *kill* text but also get it back. These commands save killed text in a *history* stack. Other commands, called *yanking* commands, retrieve elements from the history.

Deletion commands that operate on single characters do not save what they delete. However, by giving them a numeric argument, thus telling them to delete several characters, they too save the deleted text.

The commands that delete only white space do not save it.

What Histories Save

Zmacs uses several histories:

<i>Type</i>	<i>Description</i>
Kill	History of text deleted or saved. The kill history is shared with the input editor, thus allowing you to move text between files and the Lisp Listener.
Replace	History of arguments to Query Replace (<i>m-X</i>) and related commands. See the section "Searching, Replacing, and Sorting in Zmacs", page 113.
Buffer	History of editor buffers visited in this window. See the section "Manipulating Buffers and Files in Zmacs", page 129.
Pathname	History of file names that have been typed.
Command	History of editor commands that use the minibuffer, and their arguments. Commands that do not use the minibuffer, for example, <i>m-RUBOUT</i> , are not recorded in the history.
Definition	History of names of definitions that have been typed.

There is no limit to the length of a history, but the typeout window displays only the first 25 elements of the history. When the history contains more than 25 elements, the screen displays a mouse-sensitive line: *n* more elements in history. Clicking left displays the rest of the history.

Only a single instance of each of these histories exists, shared among all editors, including Zmacs, Zmail, and Dired.

Deleting Vs. Killing Text, *cont'd.*

Kill History

The kill history contains deleted text and is the history that saves the results of the commands described in this chapter. It allows you to move text from one editor window to another, for example, from the editor to a Lisp Listener. The *yanking* commands described below retrieve elements from the kill history.

Viewing the Kill History

You can view and retrieve either the complete kill history or a history of all text including a string you specify.

`c-0 c-Y`

Displays the elements of the kill history:

Kill history:

- 1: last piece of killed text
- 2: next-to-last piece of killed text
- 3: this one is a very long piece of killed text...

.
.
.

(End of history.)

Point with the mouse to any element of the history and click left to insert it into the current buffer. Only the first 25 elements of the history are shown. Click left on (*N* more elements in history.) to display the rest of the history.

`c-0 c-sh-Y`

Displays the elements of the kill history that match a string you supply. For example, if you supply "shift", you might see the following:

Kill history:

- 3: first piece of killed text to click on and shift to someplace else
- 5: Yon Cassius has a lean and shifty look
- 9: Using the shift key with yank commands adds string searching

.
.
.

(End of history.)

Deleting Vs. Killing Text, *cont'd.*

Point with the mouse to any element of the history and click left to insert it into the current buffer. Only the first 25 elements of the history are shown. Click left on (*N* more elements in history.) to display the rest of the history.

Viewing the Editor Command History

You can view and retrieve either the complete command history or a history of all commands including a string you specify.

`c-@ c-m-Y`

Displays the elements of the editor command history:

Command history:

```
1: Control-X Control-F last-file-read-in
2: Help A
3: Control-X Control-F other-file-read-in
.
.
.
```

(End of history.)

This command is context-sensitive. When typed at the Lisp Listener level, it lists the recent commands typed there. When typed at the minibuffer, it lists the history appropriate to what is being read in the minibuffer, for example, a pathname or the name of a definition.

The editor history contains only commands that use the minibuffer. Commands such as `c-N` and `c-P` are not in the history.

`c-@ c-m-Y`

Displays the elements of the editor command history that match a string you supply. For example, if you supply "sh", you might see the following:

Deleting Vs. Killing Text, cont'd.

Command history:

3: 0 Control-Meta-Shift-Y oregano
5: Show Mail
7: Show Spell Dictionaries
112: Meta-X Finish Patch

.
.
(End of history.)

This command is context-sensitive. When typed at the Lisp Listener level, it lists the matching commands typed there. When typed at the minibuffer, it lists the matching history appropriate to what is being read in the minibuffer, for example, a pathname or the name of a definition.

The editor history contains only commands that use the minibuffer. Commands such as `c-N` and `c-P` are not in the history.

**Using the Mouse
on History Elements**

History elements are mouse-sensitive. Click on an element of the kill history to yank it to point. Click on an element of the command history to reexecute the command.

Retrieving History Elements

`c-Y` Yank

Yanks back and inserts the last text killed or saved. If you have moved point since you killed the text, put point where you want the killed text to go before pressing `c-Y`. Point ends up after the text, and mark before the text. An argument of `c-U` puts point before the text instead. A numeric argument of zero displays the kill history and does not yank anything. A nonzero numeric argument selects an element of the kill history.

`c-sh-Y` Yank Matching

Yanks back and inserts the last text killed or saved that matches a string you supply. If you have moved point since you killed the text, put point where you want the killed text to go before pressing `c-Y`. Point ends up after the text, and mark before the text. An argument of `c-U` puts point before the text instead. A numeric argument of zero displays the kill history and does not yank anything. A nonzero numeric argument selects an element of the kill history.

Deleting Vs. Killing Text, *cont'd.*

`c-m-Y` Repeat Last Minibuffer Command

Repeats a recent minibuffer command. It yanks the displayed default if there is one, otherwise, it yanks the last thing typed in this context. A numeric argument n yanks the n th previous one. An argument of 0 lists the history of elements typed in the minibuffer.

For a similar command with string-matching: See the section "Repeat Last Matching Minibuffer Command" below.

`c-m-sh-Y` Repeat Last Matching Minibuffer Command

Yanks back and repeats the last minibuffer command that includes a string you specify. `m-sh-Y` yanks back previous commands that contain the same string.

`m-Y` Yank Pop

Corrects a yank to use a different element of its history. The most recent command must be a yanking command (`c-Y`, `m-Y`, `c-sh-Y`, `m-sh-Y` or `c-m-Y`). The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive `m-Y`) to bring this element to the top.

A numeric argument of zero displays the history. A positive numeric argument of n moves n elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

`m-sh-Y` Yank Pop Matching

Corrects a yank to use a different element of its history. The most recent command must be a yanking command (`c-Y`, `m-Y`, `c-sh-Y`, `m-sh-Y` or `c-m-Y`). If you supplied a matching string in the previous command, that string is used. Otherwise, `m-sh-Y` prompts for a string. The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive `m-sh-Y`) to bring this element to the top.

A numeric argument of zero displays the history. A positive

Deleting Vs. Killing Text, *cont'd.*

numeric argument of n moves n elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

Kill Merging

Normally, each kill command pushes a new block onto the kill history. However, two or more kill commands in a row combine their text into a single element on the history, so that a single `c-Y` command gets it all back as it was before it was killed. This means that you do not have to kill all the text in one command; you can keep killing line after line, or word after word, until you have killed it all, and you can still get it all back at once.

Commands that kill forward from point add onto the end of the previous killed text. Commands that kill backward from point add onto the beginning. This way, any sequence of mixed forward and backward kill commands puts all the killed text into one element without rearrangement.

If a kill command is separated from the last kill command by other commands, it starts a new element on the kill history, unless you tell it not to by saying `c-m-W` (Append Next Kill) in front of it. The `c-m-W` tells the following command, if it is a kill command, to append the text it kills to the last killed text, instead of starting a new element. With `c-m-W`, you can kill several discrete pieces of text and accumulate them to be yanked back in one place.

`c-m-W`

Append Next Kill

Makes the next kill command append text to the newest element of the kill history.

Deleting and Transposing Characters

Deleting the Last Character

RUBOUT

Rubout

Deletes the character immediately to the left of the cursor.

If the cursor is at the beginning of a line, RUBOUT deletes the newline character at the end of the previous line, thus appending the current line to the previous one.

With a positive numeric argument of n , RUBOUT deletes the n characters immediately to the left of the cursor. With a negative numeric argument of $-n$, it deletes the n characters immediately to the right of the cursor. With any numeric argument, it saves the deleted characters on the kill history.

Deleting the Current Character

c-D

Delete Forward

Deletes the character at the cursor.

If the cursor is at the end of a line, c-D deletes the newline character at the end of the line, thus appending the next line to the current one.

With a positive numeric argument of n , c-D deletes the n characters immediately to the right of cursor. With a negative numeric argument of $-n$, it deletes the n characters immediately to the left of cursor. With any numeric argument, it saves the deleted characters on the kill history.

Transposing Characters

c-T

Exchange Characters

Transposes two characters (the ones on each side of the cursor).

If the cursor is not at the end of a line, c-T transposes the character at the cursor and the character to the left of the cursor and advances the cursor one character. The result is that the character to the left of the cursor has been "dragged" one character position to the right. Repeated use of c-T continues to pull that character forward. This is useful when you are typing and enter two characters in the wrong order (for example, teh for the).

If the cursor is at the end of a line, c-T transposes the two preceding characters.

With a nonzero numeric argument of n , c-T deletes the character

Deleting and Transposing Characters, *cont'd.*

to the left of the cursor, moves forward n characters, and reinserts the deleted character. When n is negative, the cursor moves backwards.

`c-T` can only be given a numeric argument of zero when the mark is active. In this case, it exchanges the characters at point and mark.

Deleting and Transposing Words

Introduction

For a complete description of how words are delimited: See the section "Motion by Word", page 72.

Deleting the Current Word

m-D

Kill Word

Kills the word after the cursor and saves it on the kill history. If the cursor is in the middle of a word, m-D kills from the cursor to the end of that word.

With a numeric argument n , it kills n words forward from the cursor. If n is negative, it kills backward.

Deleting the Previous Word

m-RUBOUT

Backward Kill Word

Kills the word before the cursor and saves it on the kill history. If the cursor is in the middle of a word, m-RUBOUT kills from the cursor to the beginning of that word.

With a numeric argument n , it kills n words backward from the cursor. If n is negative, it kills forward.

Transposing Words

m-T

Exchange Words

Transposes the current word and the previous one. If the cursor is at the end of a line, m-T transposes the last word on that line and the first one on the next, regardless of the amount or type of white space between them.

With a nonzero numeric argument n , m-T goes to the beginning of the current word, deletes the previous word, goes forward n words, and reinserts the deleted word. Moving forward a negative amount is equivalent to moving backward. An argument of zero transposes the words at point and mark.

Deleting and Transposing Lisp Expressions

Introduction

Motion by Lisp expression repositions the cursor according to Lisp code delimiters: *lists* and *expressions*. A list is something enclosed in balanced parentheses. A Lisp expression is any readable printed representation of a Lisp object.

Deleting the Current Lisp Expression

c-m-K

Kill Sexp

Kills the Lisp expression immediately to the right of point and saves it on the kill history.

With a numeric argument of n , it kills the n succeeding expressions. It is an error to kill off the end of a containing expression. When the numeric argument is negative, it kills backwards from point the same way.

Deleting the Previous Lisp Expression

c-m-RUBOUT

Backward Kill Sexp

Kills the Lisp expression immediately to the left of point and saves it on the kill history.

With a numeric argument of n , it kills the n preceding expressions. It is an error to kill off the beginning of a containing expression. When the numeric argument is negative, it kills forward from point the same way.

Deleting the List Containing the Current Lisp Expression

Kill Backward Up List (c-m-X)

Deletes the list that contains the Lisp expression after point, but leaves that expression itself.

Transposing Lisp Expressions

c-m-T

Exchange Sexps

Point must be between two expressions to use this command.

Exchanges the two expressions on either side of point, preserving current indentation.

With a numeric argument of n , it deletes the expression to the

Deleting and Transposing Lisp Expressions, *cont'd.*

left of point, moves forward n expressions, and reinserts the deleted expression. With a negative numeric argument, it exchanges expressions in the opposite direction. An argument of zero transposes the expressions at point and mark.

Deleting and Transposing Lines

Introduction

Lines are delimited by special characters called *newlines*.

Down Line

c-N Down Real Line

Moves the cursor straight down to the corresponding column of the next line. If the cursor is positioned in the middle of the line, c-N moves it to the middle of the next one.

With a numeric argument n , it moves the cursor down n lines. Moving down a negative number of lines is the same as moving up.

Up Line

c-P Up Real Line

Moves the cursor straight up to the corresponding column of the previous line. If the cursor is positioned in the middle of the line, c-P moves it to the middle of the previous one.

With a numeric argument of n , it moves the cursor up n lines. Moving up a negative number of lines is the same as moving down.

Beginning of Line

c-R Beginning of Line

Moves the cursor to the beginning of the current line.

With a numeric argument of n , it moves the cursor to the beginning of the n th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

End of Line

c-E End Of Line

Moves the cursor to the end of the current line.

With a numeric argument of n , it moves the cursor to the end of the n th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

Deleting the Current Line

c-K Kill Line

Kills a line at a time and saves it on the kill history.

Deleting and Transposing Lines, *cont'd.*

If the cursor is at the end of a line, `c-K` kills the newline, merging the current line with the next one. If the cursor is elsewhere on the line, `c-K` kills the text between the cursor and the end of the current line.

With a numeric argument n , `c-K` kills up to the n th newline following the cursor. When n is negative or zero, `c-K` kills back to the $1-n$ th newline before the cursor. `c-0 c-K` kills from the cursor back to the beginning of the line that it is on.

Deleting Backward on the Line

`CLEAR INPUT`

Clear

Kills backward to the start of the current line and saves it on the kill history. If point is already at the beginning of the line, it kills the previous line. With a numeric argument n , it kills between point and the start of the n th line *above* the current line. Use `CLEAR INPUT` when entering a new line of text, to delete the whole line.

Transposing Lines of Text

`c-X c-T`

Exchange Lines

Exchanges the current line with the previous one and leaves the cursor at the beginning of the next line.

With a nonzero numeric argument n , `c-X c-T` deletes the previous line (including the following newline), moves down n lines, and reinserts the deleted line.

With a numeric argument of zero, `c-X c-T` exchanges the lines at point and mark, advancing both point and mark to the beginning of the next line.

Deleting Sentences

Introduction

According to Zmacs, sentences can end with question marks, periods, and exclamation points. Furthermore, these punctuation marks only end a sentence when followed by:

- A newline
- A space followed by either a newline or another space.

However, Zmacs allows any number of *closing characters*, which are ", ',), and], between the sentence-ending punctuation and the white space that follows it. A sentence also starts after a blank line.

This corresponds closely to standard typing conventions. Zmacs does not recognize a period followed by one space as the end of a sentence, for example, as in "e.g. " or "Dr. ".

Deleting the Current Sentence

m-K

Kill Sentence

Kills the text between the cursor and the end of the current sentence, and saves it on the kill history.

With a numeric argument of n , m-K kills the text between the cursor and the end of the n th sentence after the cursor, *counting* the current sentence. If the argument is negative, m-K kills $-n$ sentences *before* the cursor, counting the current sentence.

Deleting the Previous Sentence

c-X RUBOUT

Backward Kill Sentence

Kills backward one sentence and saves it on the kill history.

With a negative argument, c-X RUBOUT kills forward one sentence in a similar manner.

6. Working with Regions in Zmacs

What is a Zmacs Region?

Introduction to Regions

Many Zmacs commands deal with the region. A region consists of a block of information within the buffer that you want to manipulate as a single entity. You define the area of the region, which can be any size, from characters or chunks of code to pages or the entire buffer.

Zmacs keeps track of one or more locations in a buffer using buffer *pointers*. This section describes:

- The two buffer pointers named *point* and *mark*
- How Zmacs uses them to define the boundaries of a region
- The *point-pdl*, a ring of pointers to saved locations
- *Registers*, pointers to locations that you name and save
- The region-manipulating commands

Point and the Region

Point (shown by the cursor) is the most important buffer pointer. Most editor commands depend on the position of point. Many editor commands, invoked by either the mouse or the keyboard, can be used to position point to the desired location in the buffer. Point points to one end of the region.

Mark and the Region

Mark points to the other end of the region. To *mark* a piece of text means to position point and mark on either side of the text, making it the region. The simplest way to mark some text is to position point (using either the mouse or keystrokes) to one boundary (either the beginning or the end) of the text, set the mark there (using the Set Pop Mark command), and then reposition point at the other boundary. See the section "Setting/Popping the Mark", page 102.

Unlike point, the mark can be *active* or *inactive*. When mark is active, the region is shown on the screen by underlining. When mark is inactive, you cannot see it on the screen unless you reactivate it with `c-X c-X`. Although normally you cannot see an inactive mark, Zmacs keeps track of mark when it is inactive and sometimes uses mark in its inactive state. For example, `c-Y` leaves point and mark surrounding what it yanks, but does not activate mark. `c-W` immediately following `c-Y` kills the region even though it is not active. `c-X c-X` after `c-Y` activates mark, making the region visible. However, most commands will not use mark or the region unless it is active. You can set the mark three ways: when you create a region using the mouse, explicitly

What is a Zmacs Region?, *cont'd.*

with the command Set Pop Mark (`c-SPACE`), or with one of the commands to mark regions. See the section "Overview of Commands to Mark Regions", page 106. When you set the mark, you activate it and make the region appear.

Creating a Region

You can create a region using either the mouse or keystrokes.

Creating a Region with the Mouse

The most common way to create a region is with the mouse. Hold down the left mouse button and drag the cursor. Let up the button to mark the end of the region.

Holding down the middle mouse button creates a region, too. It marks the "thing" you point the mouse at, "thing" being mode-dependent (a word or Lisp expression if you point with the mouse at text, or a line if you point with the mouse at white space before or after all the text on the line).

Creating a Region with Keystrokes

You can also create a region using keystrokes. After setting the mark, you can move point either forward or backward to define a region in either direction; as you do so, Zmacs highlights the region with underlining.

Typing a self-inserting character or `c-G` deactivates the mark and removes the underlining that highlights the region. The mark does not have an associated cursor like point. When inactive, the mark is invisible, but you can go to it with `c-X c-X`, Swap Point And Mark.

The Point-pdl

Zmacs maintains a special stack of buffer pointers called the *point-pdl*, where *pdl* stands for *push-down list*, another name for a stack.

Zmacs automatically saves point on the point-pdl as it executes some commands (for example, `m-<`) that move point great distances. Whenever Zmacs pushes point onto the point-pdl, it displays "Point pushed" in the echo area, moves point to its new location, and pushes the previous point down onto the point-pdl.

By popping the point-pdl, that is, resetting point to its last

What is a Zmacs Region?, *cont'd.*

location as recorded on the point-pdl, Zmacs returns point to where it was when the pdl was last pushed.

Setting/Popping the Mark

c-SPACE Set Pop Mark

With no argument, c-SPACE does three things:

1. Puts mark where point is
2. Makes mark active
3. Pushes point onto the point-pdl

Other commands can do each of these operations separately. Creating a region with the mouse sets a mark and makes it active but does not push point.

This command does other things depending on how many c-U's are typed in front of it:

<i>Argument</i>	<i>Action Taken</i>
one c-U	Pops the location on the top of the point-pdl into point (typically puts point where it set the last mark).
two c-U's	Pops the location on the top of the point-pdl and throws it away.

Moving to Previous Points

c-m-SPACE Move to Previous Point

Exchanges point and top of point-pdl. With a numeric argument *n*, it rotates a ring consisting of point and the top *n-1* elements of point-pdl; thus the default argument is 2. With a numeric argument of 1, it rotates the entire point-pdl. A negative numeric argument rotates the ring in the other direction.

c-X c-m-SPACE Move to Default Previous Point

Rotates the point-pdl, the same as c-m-SPACE except that c-X c-m-SPACE has a default of 3. A numeric argument specifies the number of entries to rotate and sets the new default before rotating the point-pdl.

What is a Zmacs Region?, *cont'd.*

Showing the Mark

c-X c-X

Swap Point And Mark

Exchanges point and mark. It works even when no region is active. It highlights the text between point and mark.

With an argument, it does not exchange point and mark, but instead it highlights the text between point and mark.

Registers in Zmacs

Saving and Moving to Locations in Registers

You can assign one-character "names" to locations in the buffer, which can be helpful for setting up a series of places in your text to which you want to return for some reason – to double-check several items without interrupting your text entry or editing, if you are considering a format change that will affect several parallel points, or simply to return quickly and easily to rough spots that require further work.

c-X S Save Position

Saves the current location in a register. It prompts for a one-character register name.

c-X J Jump to Saved Position

Moves point to a position that was saved in a register. It prompts for a register name and switches buffers to move to the saved position, if necessary.

Saving and Inserting Regions in Registers

c-X X Put Register

Copies the text of the region into a register. It prompts for a register name. With a numeric argument, it deletes the region from the buffer after copying it.

c-X G Open Get Register

Inserts text from a specified register into the buffer. It prompts for the name of the register. It overwrites blank lines in the buffer the way RETURN does (using the command Insert Crs). It leaves the mark before the inserted text and point after it. With a numeric argument, it puts point before the text and the mark after.

List Registers (m-X)

Displays names and contents of all defined registers. It shows the name of the register and whether it contains a position or text. If the register contains a position, it tells which character on the line the position is at, and shows the first 50 characters on that line. If the register contains text, it shows the first 50 characters on the first line of that text.

Registers in Zmacs, cont'd.

List of all registers:

D (text) This text was marked as a region and saved here
1 (position) Char 0. in "another line containing a position"
Done.

Show Register (m-X)

Displays the contents of a register in the typeout window. It prompts for a register name and then tells whether the register contains a position or text:

Register A contains a position: Character 0 in this line:

this is the line

or

Register A contains text:

Kill Register (m-X)

Kills a register.

Commands to Mark Regions

Overview

To *mark* a piece of text means activating mark and then positioning point and mark on either side of the text, making it the region. The simplest way to mark some text is to go to one end of the text, set the mark there (using the Set Pop Mark command), and go to the other end of the text. See the section "Setting/Popping the Mark", page 102. However, several convenient commands mark different specific amounts of text:

<code>m-@</code>	Marks a word.
<code>c-m-@</code>	Marks an expression.
<code>c-m-H</code>	Marks a definition.
<code>m-H</code>	Marks a paragraph.
<code>c-X c-P</code>	Marks a page.
<code>c-X H</code>	Marks the whole buffer.
<code>c-></code>	Marks to the end of the buffer.
<code>c-<</code>	Marks to the beginning of the buffer.

Marking Words

<code>m-@</code>	Mark Word
------------------	-----------

Puts the mark at the end of the current word. With a numeric argument of *n*, `m-@` puts the mark *n* words forward from point.

Marking Lisp Expressions

<code>c-m-@</code>	Mark Sexp
--------------------	-----------

Marks the current expression by putting mark at the end.

With a numeric argument *n*, it moves forward *n* expressions and puts the mark there. For a more detailed description of how to move forward *n* expressions: See the section "Motion by Lisp Expression", page 76.

<code>c-m-H</code>	Mark Definition
--------------------	-----------------

Puts point and mark around the current definition.

Commands to Mark Regions, *cont'd.*

Marking Paragraphs`m-H`

Mark Paragraph

Puts the mark at the end of the current paragraph and moves point to the beginning, so that the current paragraph becomes the region. With a numeric argument *n*, `m-H` puts point at the beginning of the current paragraph and marks *n* paragraphs forward from there.

Example

`m-3H` marks the current paragraph and the following two; `m- -1H` marks the preceding paragraph. When marking preceding paragraphs, point is left at the end of the region, and when marking current and succeeding paragraphs, point is left at the beginning of the region.

Marking Pages`c-X c-P`

Mark Page

Puts the mark at the end of the current page and moves point to the beginning, so that the current page becomes the region.

With a numeric argument of *n*, `c-X c-P` marks the *n*th page after the current one. If *n* is zero, this is the current page; if *n* is negative, this page comes *before* the current page.

Marking Buffers`c-X H`

Mark Whole

Marks the whole buffer by putting point at the beginning and the mark at the end.

With any numeric argument, `c-X H` puts the mark at the beginning and point at the end.

Marking to End of Buffer`c->`

Mark End

Marks from the cursor to the end of the buffer by putting the mark at the end of the buffer.

Commands to Mark Regions, *cont'd.*

*Marking to
Beginning of Buffer*

c-<

Mark Beginning

Marks from the cursor to the beginning of the buffer by putting the mark at the beginning of the buffer.

Region-Manipulating Commands

Saving a Region

n-W Save Region

Puts region on kill history list without deleting it. For information on kill merging and the Append Next Kill command, c-n-W: See the section "Kill Merging", page 89.

Deleting a Region

c-W Kill Region

Deletes the region. If there is no region, c-W produces an error.

This command ignores numeric arguments and places the deleted text on the kill history list. For information on retrieving history elements and the Yank command, c-Y: See the section "Retrieving History Elements", page 87.

Compiling a Region

c-sh-C Compile Region

Compile Region (n-X)

Compiles the region, or if no region is defined, the current definition.

Transposing Regions

c-X T Exchange Regions

Exchanges two regions delimited by point and last three marks.

After transposing regions, you can undo the effect of this command by invoking it again.

Hardcopying a Region

Hardcopy Region (n-X)

Sends a region's contents to the local hardcopy device for printing.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer" in *User's Guide to Symbolics Computers*.

Filling a Region

When Zmacs *fills* text it breaks it up so that it does not extend past the *fill column*. The fill column determines the right margin, and is the first column in which text is not to be placed by n-Q, n-G, or Auto Fill Mode formatting. In addition, the *fill prefix*, if set, is inserted:

Region-Manipulating Commands, *cont'd.*

- At the beginning of each new line typed in while in Auto Fill Mode
- At the beginning of each line in a paragraph for `m-Q` and each line in a region for `m-G`

The fill prefix determines the left margin, and is empty unless set to contain some combination of spaces and characters. If you do not set the fill prefix, the left margin is the left edge of your Zmacs window. For example, to insert five spaces at the beginning of every line, insert them at the beginning of the current line, and with point at column six, use `c-X .`. To turn this fill prefix off, put point at the beginning of a line, and use `c-X .` again.

Adjusting or *justifying* text inserts extra spaces between the words to make the right margin come out exactly even.

<code>m-Q</code>	Fill Paragraph
------------------	----------------

Fills the current (or next) paragraph. A positive argument means to adjust rather than fill.

<code>m-G</code>	Fill Region
------------------	-------------

Fills the current region. A positive argument means to adjust rather than fill.

<code>c-X .</code>	Set Fill Prefix
--------------------	-----------------

Defines Fill Prefix from the current line. All of the current line up to point becomes the Fill Prefix. Fill Region starts each nonblank line with the prefix (which is ignored for filling purposes). To stop using a Fill Prefix, do a Set Fill Prefix at the beginning of a line.

Other Region-related Commands

For descriptions of the following commands:

Name and Invocation

Uppercase Region `c-X c-U`

Lowercase Region `c-X c-L`

Uppercase Code in Region (`m-X`)

Region-Manipulating Commands, *cont'd.*

Lowercase Code in Region (m-k)

See the section "Changing Case of Regions in Zmacs", page 214.

7. Searching, Replacing, and Sorting in Zmacs

Searching in Zmacs

Overview

Like other editors, Zmacs has commands for searching for an occurrence of a string. Zmacs search commands are *incremental*; that is, they begin to search as soon as you type the first character.

This section describes how to search incrementally forward and backward in the buffer, as well as a method for specifying a complete search string first and then specifying a direction in which to search.

Incremental Search

The command to search is `c-S` (Incremental Search). `c-S` reads in characters and positions the cursor at the first occurrence of the characters that you have typed. If you type `c-S` and then `t`, the cursor moves right after the first `t`. Type an `r`, and see the cursor move to after the first `tr`. Add a `y` and the cursor moves to the first `try` after the place where you started the search. At the same time, the `try` has echoed at the bottom of the screen. Stop typing when you have typed enough characters to identify the place you want.

You can rub out any character you type. After the `try`, pressing `RUBOUT` makes the `y` disappear from the bottom of the screen, leaving only `tr`. The cursor moves back to the `tr`. Rubbing out the `r` and `t` moves the cursor back to where you started the search. To exit from the search, press `END` or `ESCAPE`. You can also use `ABORT` to exit from the search. To abort out of the search and return to the original starting point in the buffer, use `c-G`.

If you want to search for something else, press `CLEAR INPUT` to erase the current search string. You are still in the search loop, so type another search string.

If the string cannot be found with `c-S`, type `c-R` to search backward for the default string. Zmacs remembers the default search string – you can reinvoke the search at any time using `c-S c-S`, to search forward for it, or `c-R c-R` to search backward.

`c-S`

Incremental Search

Searches for a character string while you type it, searching forward to the end of the buffer. It prompts for a string in the echo area with `I-Search:.` As you type characters in, `c-S` displays the accumulating string in the echo area and searches for it at the same time. If no string is found, it displays `Failing I-`

Searching in Zmacs, *cont'd.*

ESCAPE	When typed before any search characters, switches to Reverse String Search. See the section "Zmacs String Search", page 116.
END or ESCAPE	Exits the search.
c-G	Exits the search and returns to original starting point in the buffer.
c-Q	Quotes the next character, to prevent it from terminating the search.
c-S	Reverses the search to search forward.
c-R	Repeats the search.

If c-S or c-R is the first character typed, the previous search string is used again as the default. Entering any other command character terminates the search (and then executes that command).

String Search

The string search command, invoked by c-S ESCAPE, lets you type in the entire string and specify the direction in which to search before starting the search.

c-S ESCAPE String Search

Searches for a specified string, according to the arguments given with the special characters below. Another c-S always begins the search. It prompts in the echo area String Search: It saves previous string search commands on a ring, retrievable with c-D. The ring contains three elements and can be rotated with repeated c-Ds. While you are entering the search string, the following characters have special meanings:

c-B	Searches forward from the beginning of the buffer.
c-E	Searches backwards from the end of the buffer.
c-F	Leaves point at the top of the window, if the window must be recentered.
c-G	Aborts the search.
c-D	Gets a string to search for from the ring of previous search strings.

Searching in Zmacs, *cont'd.*

c-L	Redisplays the typein line.
c-Q	Quotes the next character.
c-R	Reverses the direction of the search.
c-S	Does the search, then comes back to the search command loop.
c-U or CLEAR INPUT	Erases all characters typed so far.
c-V	Delimited Search: Searches for occurrences of the string surrounded by delimiters.
c-W	Word Search: Searches for words in this sequence regardless of intervening punctuation, white space, newlines, and other delimiters.
c-Y	Appends the string on top of the string ring to the search string.
RUBOUT	Rubs out the previous character typed.
END or ESCAPE	Does the search and exits.

If you search for an empty string, it uses the default. Otherwise, the string you type becomes the default, and the default is saved unless it is a single character.

Locating and Replacing Strings Automatically

Overview of Locating and Replacing Strings Automatically

`c-?`, `Replace String`, searches forward for a string and replaces that string with another. `c-?` prompts for the string to be replaced, reads the string from the minibuffer, and then reads the replacement string. After it goes through the buffer trying to make the replacements, it tells you how many replacements it made (1. replacement.), or that it made none.

You can also substitute one string for another *selectively* throughout the buffer, with `m-?`, `Query Replace`. `m-?` prompts first for the string to be replaced (Query-replace some occurrences of:), and then for the string to replace it with (Query-replace some occurrences of "string" with:). Terminate each string you specify with RETURN. `m-?` locates each occurrence and lets you decide what to do about each one.

Making Global Replacements in Zmacs

`c-?` Replace String
 Replace String (`m-k`)

Replaces all occurrences of a given string with another, where the string can be characters, words, or phrases. It prompts first for the string to remove and second for the string to replace it with. A numeric argument *n* means to make *n* replacements. By default, it begins at point and replaces all occurrences of the first string that occur *after* point in the buffer. Usually it attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable `Case Replace P` (default `t`). When it is null, case matching does not take place. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

Querying While Making Global Replacements in Zmacs

`m-?` Query Replace
 Query Replace (`m-k`)

Starting at point, replaces a string through the rest of the buffer, asking about each occurrence, where the string can be characters, words, or phrases. It prompts for each string. You first give it

Locating and Replacing Strings Automatically, *cont'd.*

STRING1, then STRING2, and it finds the first STRING1, displaying it in context. You respond with one of the following characters:

SPACE	Replaces it with STRING2 and shows next STRING1.
RUBOUT	Leaves this STRING1, but shows next STRING1.
,	Replaces this STRING1 and shows result, waiting for a SPACE, c-R, or ESCAPE.
Period	Replaces this STRING1 and ends query replace.
c-G	Leaves this occurrence of STRING1 unchanged and terminates the query replace.
ESCAPE	Same as c-G.
^	Returns to site of previous STRING1.
c-W	Kills this STRING1 and enters recursive edit.
c-R	Enters editing mode recursively. Press END to return to Query Replace.
c-L	Redisplays screen.
!	Replaces all remaining STRING1s without asking.

Entering any other character terminates the command. Usually the command attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable Case Replace P (default t). When it is null, case matching does not take place. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

If you give a numeric argument, it does not consider STRING1s that are not bounded on both sides by delimiter characters.

Querying While Making Multiple Global Replacements

While doing multiple query replacements, you can specify the replacement strings either from the minibuffer or from another buffer altogether.

Multiple Query Replace (m-X)

Performs query replace using many pairs of strings at the same

Locating and Replacing Strings Automatically, *cont'd.*

time, where the strings can be characters, words, or phrases. (See the section "Querying While Making Global Replacements in Zmacs", page 118.) Strings are read in alternate minibuffers; when you finish entering all strings, press RETURN twice. An argument means that the strings must be surrounded by delimiter characters. A negative argument means that the strings must be delimited Lisp objects (not lists), rather than words.

Multiple Query Replace From Buffer (m-X)

Performs query replace using many pairs of strings *supplied from the specified buffer*. (See the section "Querying While Making Global Replacements in Zmacs", page 118.) The current buffer should contain a STRING1, a space, and a STRING2. Put quotation marks around any string that contains a space, tab, backspace, semicolon, or newline character. Lines in the buffer that begin with a semicolon or are blank are ignored. In other words, each string in the buffer is a Lisp string, but quotation marks can be omitted if the string contains no special characters.

Other Types of Replacement Operations in Zmacs

Besides making string replacements in text, Zmacs commands replace:

- A region into the kill history
 - Evaluated code into the buffer
 - The value of LET into its variable
 - A string for delimited Lisp objects (not lists or nil)
-

Query Replace Last Kill

Query Replace Last Kill (m-X)

Replaces the first item in the kill history with the region.

Evaluate and Replace Into Buffer

Evaluate And Replace Into Buffer (m-X)

Evaluates the Lisp object following point in the buffer and replaces it with its result.

Locating and Replacing Strings Automatically, *cont'd.*

Query Replace LET Binding

Query Replace Let Binding (m-X)

Replaces variable of LET with its value. Point must be after or within the binding to be modified.

Atom Query Replace

Atom Query Replace (m-X)

Performs query replace for delimited Lisp objects (except for lists or nil). (See the section "Querying While Making Global Replacements in Zmacs", page 118.)

Tag Tables and Search Domains

Introduction

Tag tables, a means of global searching and replacing, allow you to make sweeping changes to groups of files without having to explicitly locate each file. A *tag table* is a Zmacs support buffer, (a temporary buffer), that contains the names of sets of buffers and files, which you specify. You can edit these specified buffers and files as a unit, once you have specified them as items in a tag table. Tag tables remain active for the duration of the Zmacs session; you cannot permanently save tag tables.

You could use tag tables, for example, to:

- Search for all references to a certain variable and alter them consistently
- Search for all occurrences of an obsolete term and update it
- Search for all functions that send a certain message

How Tag Tables Work

First, you specify the buffers or files that will make up the tag table. See the section "Specifying and Listing Tag Tables", page 122. Then you can perform an operation. See the section "Performing Operations with Tag Tables", page 123. Zmacs performs the operation on the files within the tag table that you have specified.

Example

Suppose you want to perform a tag query replace in several files. Use **Tags Query Replace (m-X)** to begin: See the section "Performing Operations with Tag Tables", page 123. The minibuffer prompts as in **Query Replace (m-X)** for the string to be replaced and the replacement string. The operation begins and Zmacs displays **Control-. is now Continue query replacement of "string-old" with "string-new"**; as it displays each occurrence, you deal with each one using the appropriate response characters. **Tags Query Replace** goes through all the files specified in the tag table, listing their names in the minibuffer and stopping at each occurrence of "string-old". When it finishes searching all the files, it displays **No more files**.

Specifying and Listing Tag Tables

Select All Buffers As Tag Table (m-X)

Selects all buffers currently read in. It creates a support buffer

Tag Tables and Search Domains, *cont'd.*

called `*Tag-Table-N*`, which contains a list of the names of all the buffers. See the section "Support Buffers", page 126.

Select Some Buffers As Tag Table (m-X)

Selects some buffers currently read in, querying about each one. With a numeric argument, it asks only about buffers whose name contains a given string.

Select Tag Table (m-X)

Makes a tag table current for commands like tag search. It prompts in the minibuffer for the name of the tag table to use.

Select System As Tag Table (m-X)

Creates a tag table for all files in a system defined by `defsystem`. It prompts in the minibuffer for the name of a system – press `HELP` to see a display of system names. It selects the system but does not read the files in (use Find Files in Tag Table (m-X) to read them in).

List Tag Tables (m-X)

Lists in the typeout window the names of all the tag tables, and for each one shows the files it contains.

Performing Operations with Tag Tables

Tags Search (m-X)

Searches for the specified string within files of the tag table. It prompts in the minibuffer for the search string. If there is no current tag table, it prompts for one.

Zmacs displays in the echo area the name of each of the files in the tag table as it searches each file for the specified string. As Zmacs begins the operation and finds the first occurrence, it displays `Point pushed.` in the minibuffer and moves the cursor to the occurrence. After you deal with that occurrence, use `c-.`, the Next Possibility command, to tell locate the next occurrence. (See the section "Displaying the Next Possibility", page 126.) Go through the specified files using `c-.` to the end.

Tags Query Replace (m-X)

Replaces occurrences of one string with another within the files

Tag Tables and Search Domains, *cont'd.*

of the tag table, asking about each occurrence. It prompts first for the string to remove and second for the string to replace it with. You first give it STRING1, then STRING2, and it finds the first STRING1, displaying it in context. You respond with one of the following characters:

SPACE	Replaces it with STRING2 and shows next STRING1.
RUBOUT	Does not replace this occurrence, but shows next STRING1.
,	Replaces this STRING1 and shows result, waiting for a SPACE, c-R, or ESCAPE.
Period	Replaces this STRING1 and terminates the query replace.
c-G	Leaves this occurrence of STRING1 unchanged and terminates the query replace.
ESCAPE	Same as c-G.
^	Returns to site of previous STRING1 (actually, pops the point-pdl).
c-W	Kills this STRING1 and enters recursive edit.
c-R	Enters editing mode recursively. Press END to return to Query Replace.
c-L	Redisplays screen.
!	Replaces all remaining STRING1s without asking.

Entering any other command character terminates the command. Usually the command attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable Case Replace P (default t). When it is null, case matching does not take place. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

If you give a numeric argument, it does not consider STRING1s that are not bounded on both sides by delimiter characters.

Tags Multiple Query Replace (m-X)

Performs tags query replace using many pairs of strings at the same time, where the strings can be characters, words, or

Tag Tables and Search Domains, *cont'd.*

phrases. Strings are read in alternate minibuffers; when you finish entering all strings, press RETURN twice. An argument means that the strings must be surrounded by delimiter characters. A negative argument means that the strings must be delimited Lisp objects (excluding lists and nil), rather than words.

Tags Multiple Query Replace From Buffer (m-k)

Replaces occurrences of any number of strings with other strings within the tag table files, asking about each change. The current buffer should contain a STRING1, a space, and a STRING2. Put quotation marks around any string that contains a space, tab, backspace, semicolon, or newline character. Lines in the buffer that begin with a semicolon or are blank are ignored. In other words, each string in the buffer is a Lisp string, but quotation marks can be omitted if the string contains no special characters.

A positive numeric argument means to consider only the cases where the strings to replace occur as a word (rather than within a word). A negative numeric argument means to consider only delimited Lisp objects (excluding lists and nil), rather than words.

This command has the same options as Tags Query Replace.

Find Files in Tag Table (m-k)

Reads every file in the selected tag table into the editor. If there is no current tag table, it prompts for the name of one, which you can specify as a file (F), all editor buffers (B), or a system (S).

Visit Tag Table (m-k)

Creates a tag table by reading in a PDP-10 Emacs tag *file*. Tag files provide a list of the names of files that belong together as part of a system and a list of names and locations of definitions within the files. The file names are made into a tag table; the definition names are added to the completion table.

First, it reads in the specified tag file. It prompts for a file name from the minibuffer. Next, it goes through the tag file and marks the name of each tag as being a possible section of its file. The Edit Definition command (m-.) uses these marks to figure out which file to use.

It uses a support buffer to hold the elements of the tag table and another support buffer to hold the state of a pending operation involving all the files in the tag table. See the section "Support Buffers", page 126. Each contains the names of the files.

Tag Tables and Search Domains, *cont'd.*

Support Buffers

Zmacs creates *support buffers* to save lists that it creates as part of the execution of some commands:

- Tag table commands.
- Edit Buffers (m-X).
- View File (m-X).
- Lists for Edit Definition (m-.), when more than one definition exists.
- Buffers for Dired (m-X).
- Everything that edits a sequence of definitions, as in List Callers (m-X) or List Methods (m-X).

This means that you can examine the buffers containing the lists even after you have done some editing.

c-X c-B, the List Buffers command, displays these support buffers in the listing of buffers. Their names are, for example, *Definitions-1*, *Tags-Search-1*, and *Tags-Query-Replace-1*.

To avoid proliferation of editor buffers, Zmacs reuses support buffers in most cases, so that it usually saves no more than two of each type of support buffer at a time.

Possibility Buffers

Each time you use a command that generates a set of possibilities (for example, Tags Search (m-X) and Tags Query Replace (m-X)), it creates a possibility buffer for that set and pushes the set of possibilities onto a stack. c-., Next Possibility, extracts the next item from the set at the top of the stack. The set is popped from the stack when no more items remain in it. Several informational messages are associated with this facility. When the whole possibilities stack is empty and you have nothing more pending it displays:

No more sets of possibilities.

Displaying the Next Possibility

c-.

Next Possibility

Selects the next possibility for the current set of possibilities. With a negative argument, pops off a set of possibilities. An argument of c-U or any positive number displays the remaining possibilities in the current set. With an argument of zero, selects the current buffer of possibilities.

Tag Tables and Search Domains, *cont'd.*

For a description of the Edit Definition and Edit Callers commands: See the section "Editing Lisp Programs in Zmacs", page 223.

Example

Suppose you had been using `c-.` to move through the set provided by Tags Search and you then used Tags Query Replace to push a new set of possibilities onto the stack. When you finished the set provided by Tags Query Replace, you would see a message like the following to notify you that the empty set had been popped off the stack and the set of possibilities for Tags Search had been reinstated.:

`c-.` is now Search for next occurrence of "string"

The position of point in the support buffer indicates the next item for Next Possibility (`c-.`). You can select the support buffer and move point manually in order to skip or redo possibilities.

Typing `c-.` while in a support buffer that is not at the top of the possibilities stack moves it to the top, prints an appropriate message, then takes the next possibility from that support buffer.

Sorting

Overview

The Zmacs sorting commands alphabetically sort a region by line, paragraph, or whatever *sort key* you specify.

Zmacs Sorting Commands

Sort Lines (n-X)

Sorts the region alphabetically by lines.

Sort Paragraphs (n-X)

Sorts the region alphabetically by paragraphs.

Sort Via Keyboard Macros (n-X)

Sorts the region, prompting for actions to define the *records* (the units of the region to be rearranged) and the sort keys (the fields in the records that are compared alphabetically to determine the new order of records). It prompts you to define the records and sort keys by performing positioning commands. It prompts for three actions:

1. Move to the beginning of the sort key (that is, move the cursor to the beginning of the field upon which to sort).
2. Move to the end of the sort key (that is, move to the end of the sort field).
3. Move to the end of the sort record (that is, move to the end of the record containing that field).

For each, it records the keystrokes that you use (as keyboard macros) and plays those back to find and sort the records in the region.

8. Manipulating Buffers and Files in Zmacs

Working with Buffers and Files

Overview

Files are semipermanent collections of information stored safely outside the Zmacs environment. *Buffers*, on the other hand, are more dynamic, temporary collections of information, used by Zmacs for manipulating text. Buffers live in the active Zmacs environment. Each buffer has its own point and mark as well as other associated information.

We say we use Zmacs to "edit files", but what we really do is copy a file into a buffer created for the purpose, edit the buffer, and then write out a new version of the file from the edited buffer. The old version of the file is retained, to be deleted explicitly when appropriate. Successive versions of files are distinguished by *version number*, a component of the file name that is incremented with each new revised copy (except on file server hosts such as UNIX that do not have version numbers).

Zmacs allows multiple buffers, so that you can edit many files simultaneously. Usually only one buffer is visible on the screen at a time. You can, however, divide the screen into multiple windows so that you can view the contents of several buffers at once.

Zmacs keeps track of the association between files and buffers. If you are editing a file's contents in a buffer, Zmacs gives that buffer the same name as that of the file being edited.

Buffer and File Names

Both buffers and files have long names that indicate the host directory as well as the file name (and version, where supported). Hence completion is a necessary aid and is always provided for entering buffer and file names.

Buffer Flags for Existing Files

Each buffer has a *modification flag* that tells whether the buffer has been changed to be different from the associated file. You can see the modification flag by clicking on either the List Buffers command or the Kill or Save Buffers command in the editor menu (editor menu is click right once), or by pressing `c-X` `c-B` for List Buffers.

The modification flag is cleared when:

- The file is read into the buffer from the file system.
- The buffer is *saved*, that is, whenever its contents are written

Working with Buffers and Files, *cont'd.*

out to the associated file. As soon as its contents are modified thereafter, the modification flag is set and Zmacs displays an asterisk (*): (1) in the mode line to the right of the buffer name, and (2) whenever it displays output from the List Buffers command.

**Buffer Flags for
New Files**

The List Buffers (`c-X c-B`) command uses the plus sign (+) to mark new files that have not been saved. In addition, it uses + to mark new buffers, not associated with files, that have text in them. This helps when you put text into a new buffer and later want to be reminded to write that buffer to a file.

Selecting, Listing, and Examining Buffers

Current Buffer

At all times when using Zmacs, you have one *selected* buffer, which is the buffer that you are actively editing. This is the buffer in which all current activity takes place until you switch buffers.

Buffer History

With a single Zmacs window on the screen, the editor keeps one buffer history, the *global history list*, which remembers the previous-buffer history (stack history) of that window. The top buffer in the stack is the currently selected one. Usually, when a buffer is selected, it is pulled out of the stack and put on top. The buffers near the top are usually the most recently used. Each time you change buffers Zmacs offers the name of the most recently used buffer as the default buffer name.

When we refer to the *n*th buffer, we mean the *n*th buffer in Zmacs's stack of buffers.

Every additional window maintains its own buffer history, but the global history list continues to display an entry for every buffer in every window.

When you create a new window, Zmacs initially takes the history list for the new window from the global history list. From then on, as you switch from buffer to buffer within that window, the list for that window reflects the history of those changes in chronological order. This affects particularly `c-m-L` (Select Previous Buffer) and the default for `c-X B` (Select Buffer).

The global history list still exists and is used for name completion and `c-X c-B` (List Buffers).

Buffer Commands

Changing Buffers

`c-X B` Select Buffer

Prompts for the name of a buffer and selects that buffer, displaying its contents on the screen. If you press END or RETURN instead of a name, it reselects the second most recently selected buffer.

Using completion, it takes the string you enter and tries to complete it to an existing buffer name:

- When completion is successful, it selects that buffer.
- When completion is unsuccessful, (there is no buffer with the name given), it either waits for you to type more characters (if there are multiple possible completions) or it beeps to give you a chance to correct a typing error (if there is no possible completion). A subsequent response of `c-RETURN` creates a new buffer with the specified name and selects it.

If you precede the `c-X B` command with a numeric argument, Zmacs prompts for the name of the buffer and then creates and selects it.

`c-n-L` Select Previous Buffer

Selects a previously selected buffer. With a numeric argument *n*, it selects the *n*th previous buffer. The default argument is 2. When the argument is 1, it rotates the entire buffer history. A negative argument means to rotate the other way. An argument of zero displays the buffer history, which is mouse sensitive.

`c-X c-n-L` Select Default Previous Buffer

With a numeric argument *n*, this is exactly the same as `c-n-L`. Without a numeric argument, this command *remembers the last numeric argument it received* and uses that as its argument this time.

This is useful if you happen to be working with the top few buffers on the buffer stack and want to cycle among them without having to remember how many there are.

Listing Buffers

`c-X c-B` List Buffers

Lists all the currently existing buffers in the typeout window, along with the editor mode of the buffer and the name of the associated file, if any. For buffers with associated files, it

Buffer Commands, *cont'd.*

displays the version number of the file, if any. If there is no associated file, `c-X c-B` gives the size of the buffer in lines instead. For Dired buffers, it displays the pathname used for creating the buffer. It lists modified buffers with an asterisk. It lists the buffers sorted in stack order. You can inhibit this sorting by setting the global variable `zwei:*sort-zmacs-buffer-list*` to nil (default is t).

With an argument of `c-U`, it prompts for a substring and then lists only buffers whose names contain that substring.

The buffer names are mouse sensitive. Click right on the name of the buffer for a menu of operations (Kill, Not Modified, Save, Select) for that buffer. You can select one of the buffers by clicking left on its name.

Example

Buffers in Zmacs:

Buffer name:	File Version:	Major mode:
+ file1 /dess/zmacs VIXEN:		(Fundamental)
= *Dired-1*	VIXEN: /dess/zmacs/*	(Dired)
* doc.mss /dess/zmacs VIXEN:		(Text)
Buffer-1	[1 line]	(Fundamental)

+ means new file or non-empty non-file buffer. * means modified file.
= means read-only.

Editing Buffers

`c-m-X` Edit Buffers is not part of the standard comtab. It is similar to List Buffers (`c-X c-B`), except that the buffer listing that Edit Buffers produces is a buffer in its own right. (For an example showing how to make `c-X c-B` call Edit Buffers instead of List Buffers: See the section "Setting Editor Variables in Init Files", page 272.) It contains one line for each of the buffers in the editor.

(`c-m-X`) Edit Buffers

Displays a list of all buffers, allowing you to save or delete buffers and to select a new buffer. A set of single character subcommands lets you specify various operations for the buffers. For example, you can mark buffers to be deleted, saved, or not

Buffer Commands, cont'd.

modified. The buffer is read-only; like the Directory editor (Dired) buffer, you can move around in it by searching and with commands like `c-N` and `c-P`.

The lines in the list are not mouse sensitive. With the cursor on the line for a buffer, the following single character commands apply to that buffer:

With an argument of `c-U`, it prompts for a substring and then lists only buffers whose names contain that substring.

RUBOUT	Undeletes buffer above the cursor.
SPACE	Selects the specified buffer immediately.
D	Marks the buffer for deletion (<code>K</code> , <code>c-D</code> , <code>c-K</code> are synonyms).
U	Undeletes either the buffer on the current line or the buffer on the line above.
S	Marks the buffer for saving.
~	Marks the buffer for setting not modified.
X	Executes an extended command (same as <code>m-X</code>).

Showing a Buffer

Use Show Buffer to just look at a buffer without editing it.

<code>c-X V</code>	Show Buffer
Show Buffer (<code>m-X</code>)	

Prompts for the name of a buffer and prints out the buffer contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful, displaying a `--MORE--` message at the bottom.

SPACE, <code>c-V</code> , SCROLL	Displays the next screenful.
BACKSPACE, <code>m-V</code>	Displays the previous screenful.
RUBOUT	Exits.

Anything else exits and is executed as a command.

Buffer Commands, *cont'd.*

Inserting Command Output Into the Buffer

You might want to save some output produced by a command into the buffer, rather than seeing it displayed on the typeout window and then erased.

Execute Command Into Buffer

Execute Command Into Buffer (m-X)

Sends output from a command into the buffer. It prompts you for a command, either a key or an extended command. It inserts any typeout produced by the command into the buffer at point, rather than displaying it on the typeout window. Macro Expand Expression All (m-X) is a good example of a command whose output can be usefully saved in this manner.

Hardcopying the Buffer

Hardcopy Buffer (m-X)

Prompts for the name of a buffer and then prints the specified buffer on the local hardcopy device.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer" in *User's Guide to Symbolics Computers*.

Renaming the Buffer

Rename Buffer (m-X)

Prompts for a new name for the current buffer and changes the name accordingly. This operation removes any file association that the buffer had.

Saving Buffers

Save File Buffers (m-X)

Offers to write out each buffer that is associated with a file. It prompts in the typeout window with the name of each buffer:

```
Save file cheatin-heart.lisp >hwilliams> L: ? (Y or N) Yes.  
Save word abbrevs on file L:>hwilliams>jambalaya.qwab1? (Y or N) Yes.  
Save file rooty-tooty.text >hwilliams L: ? (Y or N)
```

Buffer Commands, *cont'd.*

Encrypting and Decrypting the Buffer

Encrypt Buffer (m-X)

Encrypts the contents of the buffer. It prompts for a key and does not echo it as you type it. It prompts for the same key again, just in case you mistyped it because of the lack of echoing, and makes sure you typed it the same both times. The encryption algorithm is the same one used by the Hermes mail-reading system.

Decrypt Buffer (m-X)

Decrypts the contents of an encrypted buffer. It prompts for a key and does not echo it as you type it. The encryption key given for decrypting must match the one used for encrypting. The encryption algorithm is the same one used by the Hermes mail-reading system.

Reading a File Into a New Buffer

Edit File (m-X)

c-X c-F

Find File

Prompts for the name of a file and looks for a buffer currently associated with that file. If one is found, it selects it. Otherwise, it creates a new buffer and reads that file into it.

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Zetalisp. See the section "Buffer and File Attributes in Zmacs", page 155.

Reading a File Into an Existing Buffer

The c-X c-V command, Visit File, is primarily useful when you type in a mistaken file name after c-X c-F and Zmacs responds (New File). You can simultaneously read in the correct file and get rid of the unwanted buffer with Visit File.

c-X c-V

Visit File

Prompts for the name of a file and reads that file into *the current*

Buffer Commands, *cont'd.*

buffer. This action associates the current buffer with the specified file.

This command can only be used if the current buffer is not already associated with an existing file.

Writing the Buffer Contents to a File

c-X c-W

Write File

Prompts for the name of a file and writes out the contents of the current buffer to the specified file. This changes the current buffer's name and associates it with the specified file.

Subsequent saves using c-X c-S save to the newly specified file. This operation clears the modification flag.

Saving the Buffer Contents to the File

c-X c-S

Save File

Writes the contents of the current buffer out to the associated file and clears the modification flag. It does not write the file if the buffer is unchanged from when the file was last visited or saved. It reads a file name from the minibuffer if the current buffer does not have an associated file.

Re-reading a File Into the Buffer

Revert Buffer (m-X)

Re-reads information into the buffer that it is associated with. For example, you can revert a Dired buffer to see the most current listing of that directory. You can also read in the most up-to-date version of a file. The command prompts for a buffer name, defaulting to the current buffer. The prompt serves as a confirmation, since Revert Buffer (m-X) throws away any modifications made to the buffer since you last saved or read the file or other information. This command is useful if you have damaged the buffer and want to start over or if the associated file is more current than the buffer. This operation clears the modification flag.

Refind File (m-X)

Re-reads a specified file into its associated buffer only if that file has changed on disk. The command prompts for a buffer name,

Buffer Commands, *cont'd.*

defaulting to the current buffer. If the associated file on disk has changed, it re-reads the file into the buffer. If the associated file on disk has not changed, it tells you that it is not necessary to re-find that file. This command is useful when more than one person works on the same program.

Refind All Files (m-X)

Re-reads only those files that have changed on disk into their associated buffers, asking about each one. If the associated file on disk has not changed, the command tells you that it is not necessary to re-find that file. This command is useful when more than one person works on the same program.

With a numeric argument, Zmacs asks you for a string, which it matches with any part of the buffer names and operates only over buffers whose names contain that string.

Creating a Fundamental Mode Buffer

Find File In Fundamental Mode (m-X)

Creates a fundamental mode buffer containing the file. This is useful because Zmacs does not parse the file while reading it in, thus the names of the functions in the file do not conflict with those already known to completion in *m-*. and similar commands. This command is necessary if the normal parsing of a Lisp Mode file signals an error, preventing it from being read into the editor to correct the cause of the error.

Associating a File with a Buffer

Set Visited File Name (m-X)

Prompts for the name of a file and associates the current buffer with that file. This command does *not* read the specified file into the buffer. Effectively, the current contents of the buffer are declared to be the new intended contents of the specified file. This command should be used with caution to avoid unintentionally destroying the old contents of the specified file.

Destroying Buffers

c-X K

Kill Buffer

Prompts for the name of a buffer and destroys that buffer. If you

Buffer Commands, *cont'd.*

press END or RETURN instead of a name, c-K destroys the current buffer and prompts for the name of a buffer to select instead.

Kill Some Buffers (m-K)

For each existing buffer, tells you something about the status of the buffer and asks whether or not to delete it. If you elect to delete a buffer that has been modified since it was last saved, the command offers to save it first.

Kill Or Save Buffers (m-K)

Puts up a multiple-choice menu listing all existing buffers. Choices are: Save, Kill, Unmodify, and Hardcopy. Specify these options next to the buffer names in the menu. This command appears on the editor menu.

Appending, Prepending, and Inserting Text

Appending a Region to a Buffer

c-X R

Append To Buffer

Prompts for the name of a buffer and appends the contents of the region onto the end of the specified buffer.

Appending a Region to a File

Append To File (m-X)

Prompts for the name of a file (Append region to end of file:) and appends the contents of the region onto the end of the specified file, writing a new version of that file.

Prepending a Region to a File

Prepend To File (m-X)

Prompts for the name of a file and prepends the contents of the region onto the beginning of the specified file.

Inserting a Buffer Into Another Buffer

Insert Buffer (m-X)

Prompts for the name of a buffer and inserts the entire contents of that buffer into the current buffer at the cursor.

Inserting a File Into a Buffer

Insert File (m-X)

Prompts for the name of a file and inserts the contents of that file into the current buffer at the cursor.

Comparing Files and Buffers

Source Compare

Source Compare (m-X)

Compares two files or buffers, prompting for type (F or B) and name of each, and displays the results of the comparison in the typeout window. It saves the output in a support buffer named *Source-Compare-N*. You can read the comparison while checking the file, for example, by going into two window mode with the comparison in one window and the file in the other.

Example

This example shows a comparison between the file new, as it was read into the buffer, and the buffer new, which contains the contents of the file new *plus* changes that have been made:

```
Source compare made by ESG on 5/21/84 12:30:40 --Fundamental--
of Buffer new /dass/pubs/pgs VIXEN: with File
VIXEN: /dass/pubs/pgs/new
```

```
****Buffer new /dass/pubs/pgs VIXEN:, Line #179
Source Compare Merge compares two files or buffers,
prompting for type and name, and merges the differences
```

```
****File VIXEN: /dass/pubs/pgs/new, Line #179
Compares two files or buffers, prompting for type and
name, and merges the differences
```

```
*****
```

```
Done.
```

Source Compare Merge

Source Compare Merge (m-X)

Compares two files or buffers, prompting for type and name, and produces a new version that reconciles the differences between the two. You choose which version (if any) to accept. You can also manually edit one or both versions.

At each place where the sources differ, the command prompts you twice. The first time you specify what to do to resolve the difference (prompts: Specify which version to keep:). (For example, you can keep one or the other version, both of them, or neither.) Respond to the prompt using these subcommands:

Comparing Files and Buffers, *cont'd.*

<i>Option</i>	<i>Action</i>
1	Leaves the first alternative in the text, redisplay the contents, and asks for confirmation of change.
2	Leaves the second alternative in the text, redisplay the contents, and asks for confirmation of change.
*	Leaves both alternatives in the text, redisplay the contents, and asks for confirmation of change.
I	Leaves both alternatives in the text, along with the message lines from the source compare (** MERGE LOSSAGE **), but does not ask for confirmation.
SPACE	Leaves both alternatives in the text, but does not redisplay the contents or ask for confirmation.
!	Disposes of this and all remaining differences the same way, without confirmation. It asks: What to do with remaining differences (1, 2, *, I, or RUBOUT? It uses whichever option you choose for the rest of the differences.
c-R	Exits from the prompt and allows you to edit. Press END to return to this question.
RUBOUT	Leaves nothing in the new buffer and does not redisplay the contents or ask for confirmation.

The second time you confirm or reject the change that was made. The screen now shows the change that was made as a result of your choice and prompts: Please confirm the change that has been made: (SPACE, RUBOUT, or c-R). Confirming it keeps that change and moves on to the next difference. Rejecting it returns to the prior appearance so that you can make a different choice:

<i>Option</i>	<i>Action</i>
SPACE	Yes, that's right.
RUBOUT	No, take that back.
c-R	Exits from the prompt and allows you to edit. Press END to return to this question.

When you finish confirming your decisions, Zmacs incorporates all changes into the new version in the specified buffer and the minibuffer displays: Done. Resectionizing the buffer.

Comparing Files and Buffers, *cont'd.*

Source Compare Merge also has a mouse interface. You can answer the first question by clicking left on the text you want to keep or on the dividing line between them to keep both. You can answer the second question by clicking left for "yes" (changes confirmed) or middle for "no" (changes rejected).

Compare/Merge

Commands for Definitions

The compare/merge commands operate on definitions by comparing, or comparing and merging, the current version with the newest version, newest version on disk, or installed version.

Comparing/Merging

Current/Newest Versions

Source Compare Newest Definition (m-X)

Compares the current definition with the newest version in the normal source file for this definition, regardless of patch files. This command never looks in patch files; it only looks in original source files. If the definition was added by a patch (so that no original source file was recorded), the command cannot find the name of the source file. However, if you read the source file into the editor, it finds the definition in the editor buffer. You can use this command for comparing patch files and source files.

Source Compare Merge Newest Definition (m-X)

Compares and merges the current definition with the newest version in the normal source file. This command never looks in patch files; it only looks in original source files. If the definition was added by a patch (so that no original source file was recorded), the command cannot find the name of the source file. However, if you read the source file into the editor, it finds the definition in the editor buffer. You can use this command for comparing patch files and source files.

Comparing/Merging

Current/Saved Versions

Source Compare Saved Definition (m-X)

Compares the current definition with the source for the newest version on disk.

Source Compare Merge Saved Definition (m-X)

Comparing Files and Buffers, *cont'd.*

Compares and merges the current definition with the source for the newest version on disk.

Comparing/Merging***Current/Installed Versions***

Source Compare Installed Definition (m-k)

Compares the current definition with the source for the installed version.

Source Compare Merge Installed Definition (m-k)

Compares the current definition with the source for the installed version, merging the results.

Window Commands

Using Two

Windows, Select Bottom

c-X 2

Two Windows

Shows two windows, selecting the bottom one. It splits the frame into two editor windows, selects the bottom one, and displays the next buffer from the global history in it. With a numeric argument, it displays that same buffer in the second window.

Using Two

Windows, Select Top

c-X 3

View Two Windows

Shows two windows, selecting the top one. It splits the frame into two editor windows, selects the top one, and displays the next buffer from the global history in it. With a numeric argument, it displays that same buffer in the second window.

Creating Two

Windows,

Specifying Other Contents

c-X 4

Modified Two Windows

Selects a buffer, file, or definition in the other window. c-X 4 combines the functions of splitting the frame and selecting contents for the second window. It prompts for the type of contents you want for the second window: Select what in other window? (B, F, D, or J), for buffer, file, definition, or jump to register. Then it reads the name of the file, buffer, definition, or register that you want to select for that window.

Creating Two

Windows with the

Region in Top

c-X 8

Two Windows Showing Region

Makes two windows on the same buffer, with the top one displaying the current region.

Changing Window Size

c-X ^

Grow Window

Changes the size of the current window by some number of lines. With a positive numeric argument, it expands the window; with a negative numeric argument, it shrinks the window.

Window Commands, *cont'd.*

Choosing the Other Window

`c-X 0` Other Window
 Moves the cursor to the other window.

Returning to One Window

`c-X 1` One Window
 Returns the editor frame to displaying only one window. It expands the current window to use the whole frame. With a numeric argument, it expands the other window to use the whole frame.

Scrolling the Other Window

`c-m-U` Scroll Other Window
 Scrolls the other window up several lines. By default, it scrolls the same way as `c-U`. With no argument, it scrolls a full screen. With just a minus sign as an argument (`c-m- -U`), it scrolls a full screen backward. A numeric argument tells it how many lines to scroll – a positive number scrolls forward, a negative number scrolls backward.

Splitting the Screen

Split Screen (`m-X`)
 Pops up a menu that offers to create a new buffer or find a file; makes several windows split among the buffers as specified.

File Manipulation Commands

Overview

The commands described in this section are unlike most other Zmacs commands. Their main business is not manipulating buffers and their contents, but rather files out in a file system. First we discuss some commands for dealing with files, then we describe buffer and file attributes, and finally we explain *Dired Mode*, a special Zmacs mode for directory editing.

Creating a Directory

Create Directory (m-X)

Creates a new directory. It prompts for a directory name, using the standard conventions for defaults. For consistency between hierarchical and nonhierarchical file systems, you specify the directory to be created as the directory component of a pathname. That is, you must end the directory name with whatever delimiter or separator is appropriate for the host.

Example

<i>Host</i>	<i>Directory string</i>	<i>Result</i>
TOPS-20	<A.B.C>	Creates directory C
Multics	>udd>Sun>Luna>z>	Creates directory z
Lisp Machine	>sun>luna>b>	Creates directory b
UNIX	/usr/jek/new/	Creates directory new

Currently, the file servers for VAX/VMS and TOPS-20 can fail to create directories, due to missing options.

Listing Files in a Directory

List Files (m-X)

Prompts for the name of a directory and displays the names of all the files in that directory.

The file names are mouse sensitive. Pointing at a file name and clicking left is the same as doing a c-X c-F (Find File) on that file. Clicking right pops up a menu with three items:

Load	Loads the file into the Lisp world. The file must be either a Lisp source file or a compiled Lisp (<i>bin</i>) file.
Find	Reads the file into an editor buffer.
Compare	Compares the file with its most recent version and prints the differences.

File Manipulation Commands, *cont'd.*

Displaying the Contents of a Directory

c-X c-D

Display Directory

Displays the directory of the file in the current Zmacs buffer. **c-X c-D** does not ask for a directory but lists files with the same host, device, directory, and name as the file in the current buffer. It lists files with any type and version. With a numeric argument, it prompts for a directory to list and lists that directory.

The heading of the directory listing is mouse sensitive; clicking left on it selects a Dired buffer containing that directory listing.

c-U c-X c-D does the same thing as List Files, except that it gives more details about each file.

Show Directory

Show Directory (m-X)

Prompts for the name of a directory and displays the directory contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful displaying a --MORE-- message at the bottom.

SPACE Displays the next screenful.
 BACKSPACE Displays the previous screenful.
 RUBOUT Exits.

Anything else exits and is executed as a command.

Show Login Directory

Show Login Directory (m-X)

Displays the directory contents of the user's home directory for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful displaying a --MORE-- message at the bottom.

SPACE Displays the next screenful.
 BACKSPACE Displays the previous screenful.
 RUBOUT Exits.

Anything else exits and is executed as a command.

File Manipulation Commands, *cont'd.*

Showing a File

Use Show File to look at a file without editing it.

Show File (m-K)

Prompts for the name of a file and displays the file contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screenful, displaying a --MORE-- message at the bottom.

SPACE, c-U, SCROLL Displays the next screenful.

BACKSPACE, m-V Displays the previous screenful.

RUBOUT Exits.

Anything else exits and is executed as a command.

Showing the Properties of a File

Show File Properties (m-K)

Prompts for the name of a file and displays all the properties of the file that are maintained by the file system on which it resides. These are the properties such as creation date and time, author, time of last access, and length. For files on a Lisp Machine file system, it displays user-defined properties as well.

It prompts for a file specification, which it merges with the current default to form the pathname. Wildcards are not accepted; this must correspond to a unique file or directory name.

Changing the Properties of a File

Change File Properties (m-K)

Edits the properties of a file. Properties are the qualities of the file that are maintained by the file system on which it resides, such as creation date and time, author, time of last access, and length. For files on a Lisp Machine file system, this means user-defined properties as well. It prompts for the name of a file and pops up a choose-variable-values window, allowing you to alter various properties of the file. The exact properties that can be altered depend on the file system, but they might include:

File Manipulation Commands, *cont'd.*

- Generation (version) retention count
 - Author
 - Creation, modification, and reference dates
 - Protection flags
 - Other file-associated information
-

Hardcopying a File

Hardcopy File (n-X)

Prompts for the name of a file and then prints the specified file on the local hardcopy device.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer" in *User's Guide to Symbolics Computers*.

Renaming a File

Rename File (n-X)

Renames one or more files. It prompts for the name of a file and then asks for a new name for that file. It renames the specified file with that new name.

If the source file specification is wild, the target file specification must also be wild.

Copying a File Into Another

Copy File (n-X)

Copies any type of file to another specified file.

Prompts from the minibuffer for the names of two files and copies the contents of the first into the second. In file systems supporting multiple versions, this creates a new version of the second file whose contents are identical to those of the first.

Copy File determines whether the source file is a character file or a binary file and copies the file appropriately. Different file systems sometimes use different character sets, and if the file is a character file, character translations have to be done (for example, on some hosts Return characters have to be converted into a carriage return and a line feed).

The numeric argument controls copying of attributes and properties. With no numeric argument, it copies creation date and author and determines the mode (binary or character) of copy by the file being copied. To force mode, or suppress author or

File Manipulation Commands, *cont'd.*

creation date copying, supply a numeric argument created by adding the values corresponding to the descriptions below:

- 1 Force copy in 16-bit binary mode.
- 2 Force copy in character (text) mode.
- 4 Suppress copy of author.
- 8 Suppress copy of creation date.

Examples

For example, to suppress author and creation date for copying:

```
c-12 Copy File (m-X)
```

Use wildcard pathnames to specify groups of files for copying.

For example, to copy all files in the subdirectory mine:

```
F:>program>mine>*.*
```

If the source file specification is wild, the target file specification must also be wild.

```
you type: m-X Copy File
Zmacs: Copy File from:
you type: src:<lmfs>*.l*sp;0
(Copies all the newest .LISP and .LSPs)
Zmacs: to:
you type: ff:>sys-hold>src-sources>old-*.*.
Zmacs: SCRC:<LMFS>TEST.LSP.3 is copied into
ff:>sys-hold>src-sources>old-test.lisp.3

SCRC:<LMFS>FILES.LISP.147 is copied into
ff:>sys-hold>src-sources>old-files.lisp.147
```

Note that .LSP gets mapped into .lisp because Copy File uses canonical types when the type of the target pattern is :wild.

This command can copy file authors and creation dates, when the target operating system supports setting these attributes. This action is not the default.

Creating Links to Files

Create Link (m-X)

Creates a link to a file. It prompts in the minibuffer for the names of two files as arguments; first the name of the link, then the name of the target pointed to by the link.

File Manipulation Commands, *cont'd.*

Deleting Files

Delete File (m-X)

Deletes a file. It prompts in the minibuffer for a file name, which can be wild. With a wild name as an argument, deletes multiple files. It lists the files that would be deleted and requires that you confirm the list. It deletes the files, showing any errors that occur but continuing rather than halting. Displays a message in the minibuffer if the specified file does not exist.

Deleting Multiple Versions

Reap File (m-X)

This command works in file systems supporting multiple versions. It prompts for the name of a file (not including version number) and deletes excess or temporary versions of the specified file, keeping the most recent *n* files. Any numeric argument specifies the number of versions to keep. With no numeric argument, the default keeps two versions and deletes any excess. It prompts for confirmation of files being deleted.

Note:

- To specify file types to be automatically marked for deletion, change the value of the variable `zwei:*temp-file-type-list*`, which contains a list of these files. Its default values are: "memo", "xgp", "@xgp", "unfasl", "output", "olrec" and "press". This variable also accepts the value `:anything`, which can be any file type.
- To alter the default number (2) of versions to be kept, change the value of the variable `zwei:*file-versions-kept*` to any `:fixnum`.

Clean Directory (m-X)

Deletes excess versions or temporary file types in the specified directory. The default for excess versions is more than two. It prompts for confirmation of files being deleted. With a numeric argument *n*, it deletes excess versions greater than *n*.

Excess is defined by the value of the Zmacs variable File Versions Kept or by the numeric argument. The temporary file types are defined by the Zmacs variable Temp File Type List. It accepts wildcards in the file name specification. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

File Manipulation Commands, *cont'd.*

Buffer and File Attributes

Attributes

Each buffer and generic pathname has *attributes*, such as Package and Base, which can also be displayed in the text of the buffer or file as an attribute list. An attribute list must be the first nonblank line of a file, and it must set off the listing of attributes on each side with the characters `-*-`. If this line appears in a file, the attributes it specifies are bound to the values in the attribute list when you read or load the file.

How They Work

Suppose you want your new program to be part of a package named `graphics` that contains graphics programs. In this case, you want to set the Package attribute to `graphics` in three places: the generic pathname's property list; the buffer data structure; and the buffer text. Here are two ways to make the change:

- If the package already exists in your Lisp environment, use Set Package (`m-X`) to set the package for the buffer. The command asks you whether or not to set the package for the file and attribute list as well. You can use this command to create a new package.
 - Use Update Attribute List (`m-X`) to transfer the current buffer attributes to the file and create a text attribute list. Edit the attribute list, changing the package. Use Reparse Attribute List (`m-X`) to transfer the attributes in the attribute list to the file and the buffer data structure. If the package you specify by editing the attribute list does not exist in your Lisp environment, Reparse Attribute List asks you whether or not to create it with default characteristics.
-

Attribute-Manipulating Commands

Update Attribute List (`m-X`)

Updates the attribute list (`-*-` line) of the buffer. It creates or updates the attribute list of the file, using the current set of parameters. A new attribute list inherits the default base (10) and the default syntax (Common-Lisp) plus the Package, Mode, Backspace, and Fonts attributes of the current buffer. It includes the Backspace and Fonts attributes in the line only if they have values other than the defaults. It does not change other attributes in an existing mode line.

Buffer and File Attributes, *cont'd.*

Reparse Attribute List (m-X)

Reparses the attribute list (-*- line) of the buffer. It finds the attribute list for the buffer and processes it to set up the environment that the line specifies. It changes the major mode, package, base, and so on, as necessary. When you edit the attribute list, you should then use this command to make the changes take effect in Zmacs. The changes take effect both for the editor buffer and for the file that the buffer is editing.

Example

Suppose the package for the current buffer is **user** and the base is 8. You want to create a package called **graphics** for the buffer and associated file. You also want to set the base to 10. If no attribute list exists, use Update Attribute List (m-X) to create one using the attributes of the current buffer. An attribute list appears as the first line of the buffer:

```
;;; -*- Mode: LISP; Package: USER; Base: 8 -*-
```

Now edit the buffer attribute list to change the package name from **USER** to **GRAPHICS** and to change the base from 8 to 10. Use Reparse Attribute List (m-X). The command queries:

```
The file belongs in package GRAPHICS, which does not exist.
Create it with default characteristics,
  Try again, or Use another package? (C, T, or U)
```

Answer C to create the new package. The package becomes **graphics** and the base 10 for the buffer and the file.

File Attribute Checking

Zmacs notes errors in file attribute lists and warns you when it finds an unknown attribute. It goes ahead and ignores the unknown attribute in the list. The purpose of the warning is simply to help you detect misspellings.

Setting the Package

Set Package (m-X)

Changes the package associated with the buffer. It prompts for a new package, offering to create the package if necessary. Forms that are read from the buffer are read in that package. (The default value for this attribute is **user**.)

You can have any package as the default package by specifying it as the value of the Zmacs variable **Default Package**. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.) You can set the variable in

Buffer and File Attributes, *cont'd.*

your init file by using the internal form of its name. (See the section "Creating an Init File", page 272.)

For example, in your init file:

```
(login-forms
  (setf zwei:*default-package* (pkg-find-package "tv")))
```

If you set the variable to `nil`, it sets the default to the package from the previous buffer.

Information about the package attribute exists in four places. Set Package offers to set the package for the generic pathname attribute list and updates the attribute line in the buffer when you answer Yes to:

Set it for the file and attribute list too?

Your answer affects the various versions of the package attribute as follows:

<i>Location</i>	<i>"Y"</i>	<i>"N"</i>
Generic pathname	changes	same
Buffer property	changes	changes
Buffer text	changes	same
Current package	changes	changes

The system is informed that the file belongs to the specified package. If you are not sure what to answer, say Yes. The global variable `zwei:*set-attribute-updates-list*` controls this query. Its default value is `:ask`. Setting the variable to `t` means Yes; `nil` means No.

Base and Syntax Defaults

The default value of `zl:base` and `zl:ibase` is 10. If you have been writing code that has a Base attribute in the mode line, you should not experience any difficulties. However, in order to help avoid problems in general, changes have also been made to the editor and compiler:

- In the mode line (the `-*-` line in Lisp source files) are the Base and Syntax attributes. The base can be either 8 or 10 (default). The syntax of a program can be either Zetalisp or Common-Lisp.
- If there is a Base attribute, but no Syntax attribute, the syntax defaults to Common-Lisp.

Buffer and File Attributes, *cont'd.*

- If there is a Syntax attribute of Common-Lisp, and no Base attribute, the base is assumed to be 10.
 - If there is neither a Base nor a Syntax attribute, Base is assumed to be the default base (10) and the syntax is assumed to be Common-Lisp. Furthermore, a warning is issued to the effect that there is neither a Syntax nor a Base attribute. You should edit your program accordingly. With most programs, the Zmacs command Update Attribute List (*m-X*) will add the appropriate attributes to the mode line, following the above defaults.
-

Setting the Syntax for Symbolics Common Lisp

If you use the new Symbolics Common Lisp (SCL), you must explicitly set the syntax in the file attribute line (formerly, Zetalisp was the implicit default). For more information about SCL: See the section "Introduction to Symbolics Common Lisp" in *Symbolics Common Lisp*.

The file attribute line of a Common Lisp file should be used to tell the editor, the compiler, and other programs that the file contains a Common Lisp program. The following file attributes are relevant:

Syntax	The value of this attribute can be Common-Lisp or Zetalisp. It controls the binding of the Zetalisp variable <code>zl:readtable</code> , which is known as <code>*readtable*</code> in Common Lisp. The default syntax is Common-Lisp.
Package	<code>user</code> is the package most commonly used for Common Lisp programs. You can also create your own package. Note that the Package file attribute accepts relative package names, which means that you can specify <code>user</code> rather than <code>cl-user</code> .

The following example shows the attributes that should be in an SCL file's attribute line:

```
;;; -*- Mode:Lisp; Syntax:Common-Lisp; Package:USER -*-
```

Buffer and File Attributes, *cont'd.*

Set Lisp Syntax (m-X)

Changes the buffer into Common-Lisp syntax or Zetalisp syntax. It asks whether to update the attribute list (-*- line) of the buffer. If you answer yes, it creates or updates the attribute list of the file, using the current set of parameters, if any. It does not change other attributes in an existing mode line.

Other Set

Commands for File and Buffer Attributes

Each of the file attributes has a Set command associated with it. You have two choices when you want to change an attribute for a file:

- Edit the text of the buffer and then use Reparse Attribute List.
 - Use the relevant Set command and answer Y to its query. The meanings for Y and N are the same as for the Set Package command (except that only the Set Package command affects the current package).
-

Update Attribute List Query

The Set commands use the value of the global variable `zwei:set-attribute-updates-list*` to determine whether to query you about updating the file attribute list. The default value for the variable is `:ask`; set to `nil` to suppress the query.

<i>Value</i>	<i>Meaning</i>
<code>:ask</code>	Always asks whether to update the attribute list.
<code>nil</code>	Never updates the attribute list.
<code>t</code>	Always updates the attribute list.

Set *attribute* (m-X)

where *attribute* is one of the following: Backspace, Base, Fonts, Key, Lowercase, Nofill, Package, Patch File, Syntax, Tab Width, Variable, or Vsp. It sets *attribute* for the current buffer. It queries whether or not to set *attribute* for the file and in the text attribute list.

Attribute Descriptions

The following table describes some of the attributes, their associated Set commands, and the default value for the attribute.

Buffer and File Attributes, cont'd.

Backspace The Set Backspace command (default value **nil**) controls whether a backspace character in a file displays as the word "back-space" with a lozenge around it or performs the backspace. The default is the lozenge form.

Base The Set Base command (default value 10) specifies the value of **zl:ibase** that the Lisp reader uses when reading forms from the file. Thus, Base controls the **zl:ibase** used when you evaluate or compile parts of the buffer, *and* controls the value of **zl:base** for printing during evaluating all or part of the buffer. This value does not affect the values of either **zl:base** or **zl:ibase** in the Lisp Listener you get by using **SUSPEND**.

Fonts The Set Fonts command (default value **nil**) changes the set of fonts to use. It reads a sequence of font names separated by spaces, commas, or both from the minibuffer.

Lowercase The Set Lowercase command (default value **nil**) means that the file being edited is intended to contain lowercase code or text. When the Lowercase attribute is **nil** (that is, not present), whatever case handling you specify prevails. To automatically uppercase code, use the following in your init file:

```
((login-forms
  (setf zwei:lisp-mode-hook
    'zwei:electric-shift-lock-if-appropriate))
```

(See the section "Creating an Init File", page 272.) When the Lowercase attribute is anything but **nil** (you answer **Y** to its query), the Electric Shift Lock Mode is never turned on automatically.

Buffer and File Attributes, cont'd.

Nofill	<p>The Set Nofill command has a default value of <code>nil</code>, which means that whatever autofilling behavior you specify prevails. When Nofill is anything else (you answer <code>Y</code> to its query), it means that autofilling is not appropriate for people who specify the mode of "autofilling if appropriate".</p> <p>Use Nofill sparingly. Setting it means that everyone who edits the file has to be satisfied with Auto Fill Mode being off by default. In most cases, it is more reasonable to let an individual user's preferences prevail. It is useful for files that are not plain text, such as mailing lists, where you need to avoid spurious line breaks.</p> <p>To have autofilling turned on by default, use the following in your init file (See the section "Creating an Init File", page 272.):</p> <pre>(login-forms (setf zwei:text-mode-hook 'zwei:auto-fill-if-appropriate))</pre> <p>People who do not want it never get it by default.</p>
Patch-File	<hr/> <p>The Set Patch File command has a default value of <code>nil</code>, which means that the file does not contain patches. When a file is classified as containing patches (you answer <code>Y</code> to its query), <code>fdefine</code> does not warn about functions being redefined during loading. Classifying something as a patch file also affects Edit Definition (which prefers files that are not patches) and <code>defvar</code> (which becomes <code>zl:setf</code>).</p>
Tab-Width	<hr/> <p>The Set Tab Width command (default 8 characters) specifies how many spaces the editor uses between "tab stops".</p> <hr/>

Buffer and File Attributes, *cont'd.*

Vsp

The Set Vsp command (default 2 pixels) specifies the vertical spacing (in pixels) between the text lines of an editor window. It specifies the distance between the descenders of one line and the ascenders of the next.

Dired Mode

Overview

There is a special Zmacs mode, called *Dired*, just for doing housekeeping in a directory. In this mode, you see the names of all the files in a directory at once, and can manipulate these files in various ways.

Entering Dired

The following commands specify a directory to manipulate and enter Dired mode.

Dired (m-X)

Edit Directory (m-X)

Prompts for a wildcard file specification for files contained in the specified directory. The default edits all files in the current directory by specifying wild name, type, and version. You must type the pathname in the form acceptable to your host system.

c-X D

Dired

Edits the files in the directory that contains the current file.

With a numeric argument of 1, shows files with the same host, device, directory, and name as the file in the current buffer. It lists files with any type and version.

With a c-U argument, it prompts for a wildcard file specification showing the name of a directory to edit.

The Dired Display

When you go into Dired mode, Zmacs creates a special buffer that contains the names of the files that are under consideration, as well as some auxiliary information pertaining to those files. In a typical Dired buffer, each line describes a single file and lists the following information, from left to right:

- An indicator (D) that shows if the file has been marked for deletion or is already deleted
- The physical volume of the file (on some hosts)
- The name of the file
- The length of the file in blocks (where the length of a block is system-dependent)
- The length of the file in bytes, followed by the byte length in bits, enclosed in parentheses
- ! if the file has not been backed up to tape

Dired Mode, *cont'd.*

- \$ if the file has been marked against reaping
- @ if the file has been marked against deletion
- The file's creation date
- The file's creation time
- The date the file was last referenced, enclosed in parentheses
- The author of the file
- Optionally, the name of the last user to read the file

If there are too many files to be displayed in one screenful, the Zmacs window looks only at one section of the directory at a time (although the buffer does contain the names of all the files).

The files are arranged in alphabetical order by name.

Updating the Display

Use the Revert Buffer ($m-X$) command to update a Dired display. (See the section "Re-reading a File Into the Buffer", page 138.) After using Dired commands (or native host commands) to perform operations on files in your directory, invoke Revert Buffer, which reexecutes Dired with the default directory name and re-reads the updated directory into the buffer.

Dired Commands

Dired mode has its own command table (comtab) for manipulating the files whose names are displayed. These commands are described in this section. All invocations given in this section are with respect to the Dired comtab and do not apply to regular Zmacs.

You use Dired by moving the cursor around to various lines and then specifying operations to be performed on the file listed on that line (the *current file*, while in Dired Mode).

Most Dired commands schedule some action for the future rather than performing it instantly. For example, when you want to delete a file using Dired, you move the cursor to the line describing that file and type D. Rather than deleting the file immediately, Dired *marks the file for deletion*. The deletion actually happens when you leave Dired mode and confirm your request. (See the section "Getting Out of Dired", page 166.)

Some of the commands in Dired mode take numeric arguments. You type numeric arguments in exactly the same way as you do in Zmacs proper, except that you do not have to hold a modifier key down while typing the argument – just typing the number suffices.

Dired Mode, cont'd.

Command Summary

The following table summarizes the Dired commands:

<i>Character</i>	<i>Action</i>
RUBOUT	Undeletes file above the cursor.
SPACE	Moves to the next file.
	Moves to the next file that is not backed up.
\$	Complements the Don't Reap (\$) flag.
,	Describes the attribute list of this file. In text files, this is the <i>-*</i> line of the file. In compiled Lisp files, it includes information about the compilation as well.
.	Changes properties of current file.
@	Complements the Don't Delete (@) flag.
=	Compares this file with the newest version (Source Compare).
R	Queues this file for function application.
C	Copies this file to someplace else.
D	Marks the file for deletion (K, c-D, c-K are synonyms).
E	Edits the file in a buffer, or runs Dired if the line is a subdirectory name.
G	Sets and enforces the generation retention count.
nH	Marks excess versions of the file for deletion (argument means whole directory).
L	Loads the file into Lisp.
nN	Moves to the next file with more than <i>n</i> versions (see the Zmacs variable File Versions Kept). (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)
P	Prints the file on the standard hardcopy device.

Dired Mode, cont'd.

- Q** Exits. It shows the files marked for deletion and prompts for confirmation. The exit display marks files that have special status, using the following marks:
- : a link
 - > most recent version
 - \$ file marked for not reaping
 - ! file not backed up
- R** Renames this file to something else.
- U** Undeletes either the file on the current line or the file on the line above.
- V** Views the file without creating a buffer (using View File conventions).
- X** Executes an extended command (same as $m-X$).
-

**Default Pathnames
in Dired**

When the current buffer is a Dired buffer, and you execute an editor command that accepts a file name as an argument, the default file name is the file name that appears on the line of the Dired buffer that point is on.

This makes it easier to do things to the file that you are currently operating on in Dired. For example, you can move point to some line, do Compile File ($m-X$), and the command defaults to that file name.

Getting Out of Dired

Q Dired Exit
END

Leaves Dired mode. It prints the names of files marked for various actions and gets your final confirmation that these actions are really to be performed.

At this point the available options are:

- Y** Delete but do not expunge, also doing any other marked actions.
- N** Go back to Dired.

Dired Mode, *cont'd.*

Loading a File in Dired

L	Load File
---	-----------

Loads the current file. It displays a message Loading the file... in a typeout window and finishes with the message Loading done.

Moving Around in Dired

SPACE	Down Real Line
-------	----------------

c-N

Moves point to the next line (same as in regular Zmacs). With a numeric argument of n , it moves point forward n lines.

c-P	Up Real Line
-----	--------------

Moves point to the previous line (same as in regular Zmacs). With a numeric argument of n , it moves point backward n lines.

Viewing File Attributes in Dired

,	Dired Describe Attribute List
---	-------------------------------

This command is also available on the pop-up menu that you get when you click right in Dired. It prints out the contents of the attribute list of the current file (the one where point is). It works for character files and compiled files.

Changing File Properties in Dired

.	Dired Change File Properties
---	------------------------------

This command is also available on the pop-up menu that you get when you click right in Dired. It edits the properties of the current file. These properties are the qualities of the file that are maintained by the file system on which it resides, such as creation date and time, author, time of last access, and length. For files on a Lisp Machine file system, this means user-defined properties as well. It pops up a choose-variable-values window, allowing you to alter various properties of the file. The exact properties that can be varied depend on the file system, but they might include:

- Generation (version) retention count
- Author

Dired Mode, *cont'd.*

- Creation, modification, and reference dates
 - Protection flags
 - Other file-associated information
-

Viewing and Editing File

Contents in Dired

You might want to look at the contents of a file before deciding what to do with it. You might also want to read the file into a buffer and edit it.

V Dired View File

Displays the contents of the current file on the typeout window.

Use this command when you just want to skim the contents of the file, not edit it. You can move forward while viewing with SPACE, c-V, or SCROLL and move backward with BACKSPACE or m-V.

E Dired Edit File

Reads the current file into a Zmacs buffer and selects that buffer. You are then back in normal Zmacs and can edit the file normally. When you want to return to Dired mode, just use the c-m-L command to reselect the Dired buffer.

Comparing Recent Versions of Files in Dired

Often before deciding whether or not to delete a file, you want to find out exactly how extensive the differences are between the file and its most current version.

= Dired Srccom

Compares the current file with its most recent version and displays the differences on the typeout window. With an argument of c-U, it asks what version to compare it to.

Copying and Renaming Files

C Dired Copy File

Copies the current file. It prompts for the new pathname, displaying the default pathname.

Dired Mode, *cont'd.*

R Dired Rename File

Renames the current file. It prompts for the new pathname, displaying the default pathname.

Marking Files for Deletion

D Dired Delete

K

c-D

c-K

Marks the current file for deletion. Dired puts a D in the first column to show that the file has been so marked.

With a numeric argument of n , it marks the next n files for deletion.

Sometimes you mark a file for deletion by mistake. Here is how you recover from this error:

U Dired Undelete

U takes one of two actions:

1. If the current file is marked for deletion, printing, or a function application (with a D, P, or A), reprints it.
2. In file systems with soft deletion, U marks a deleted file for undeletion.

In either case, U removes the D, P, or A next to the file. If the current file is not marked with D, P, or A, U reprints the file on the immediately preceding line, positioning point on that line.

With a numeric argument of n , it reprints the files on the next n lines including the current line.

RUBOUT Dired Reverse Undelete

Reprints the file on the preceding line.

With a numeric argument of n , it reprints the files on the previous n lines including the current line.

Deleting Multiple Versions

If you are using Dired for housekeeping purposes, the following commands are useful:

Dired Mode, cont'd.

N Dired Next Hog

Moves point to the next file with superfluous versions. Superfluous is defined by the value of the Zmacs variable File Versions Kept (whose default is 2) or by a numeric argument. (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 269.)

H Dired Automatic

This command is also available on the pop-up menu that you get when you click right in Dired. It marks all the superfluous versions of the current file for deletion. With an argument of c-U, it marks superfluous versions of all files in the Dired buffer.

Setting Generation**Retention Count**

G Dired Set Generation Retention Count

Sets and enforces the generation retention count on this group of files, which specifies how many versions to save (that is, deletes multiple versions).

With a numeric argument *n*, sets it to *n* versions. With no numeric argument, prompts for a number in the minibuffer. An argument of zero means save all versions. *Enforce* means mark for deletion or undeletion.

Protecting Files**From Being Reaped**

In addition to keeping other users aware of protected files, protection features can also inform the system itself. Some file systems have automatic reaping facilities that go into action when storage becomes scarce. Most such systems have a *don't reap* bit associated with each file; use it to protect only your most vital files.

\$ Dired Complement No Reap Flag

Complements the Don't Reap flag associated with the current file; Dired displays the flag as \$ between the length and date on that line. With a numeric argument of *n*, it complements the flag on the next *n* files, including the current one.

Protecting Files**From Being Deleted**

@ Dired Complement Dont Delete Flag

Dired Mode, *cont'd.*

Complements the Don't Delete flag associated with the current file; Dired displays the flag as @ between the length and date on that line.

With a numeric argument of n , it complements the flag on the next n files, including the current one.

Finding Files That Have Not Been Backed up

Many file systems have tape backup facilities so that files can be copied onto tape against the possibility of a file system disaster. These systems almost always associate a bit with each file that is set when the file is created or modified and cleared when it is backed up to tape.

! Dired Next Undumped

Moves point forward to the next file that has not yet been backed up; Dired displays the flag as ! between the length and date on that line.

Marking Files to Be Hardcopied

You might want to obtain a hardcopy of a group of related files. Dired allows you to mark files to be hardcopied as well as to be deleted.

P Dired Hardcopy File

Marks the current file for printing. Dired puts a P in the first column to show that the file has been so marked.

With a numeric argument n , marks the next n files for printing.

Applying Arbitrary Functions to Files

Very occasionally, you want to perform some operation on selected files in your directory for which there is no Dired command provided. When this occurs, you can write up the operation that you want to perform as a Lisp function, whose single argument is the pathname of the file. The following command is relevant:

A Dired Apply Function

Marks the current file for having an arbitrary function applied to

Dired Mode, *cont'd.*

it. Dired puts a A in the first column to show that the file has been so marked. With a numeric argument of n , it marks the next n files, including the current one.

9. Setting the Zmacs Major Mode

Major Editing Modes

Overview

Whenever you are editing some text, some set of modes is in effect. The buffer is always associated with one major mode that tells the editor what kind of document is being edited. A major mode has the following characteristics:

- It has its own distinct set of key bindings.
- It affects groups of related language-specific items, such as delimiter characters and indentation rules.

The major modes are listed below. You can establish the mode:

- By turning it on using the prefix `m-X` followed by the name of the mode. For example, to invoke Lisp Mode, type: `m-X Lisp Mode`.
- By setting it in the attribute list. See the section "Buffer and File Attributes in Zmacs", page 155.
- By having Zmacs do it for you when you specify a file with `c-X c-F` or the Edit File command. It recognizes the type component of the pathname of the file (for example, `folon.lisp`) and puts the buffer in the corresponding mode.

Fundamental Mode

Fundamental Mode enters Zwei's fundamental mode (the default mode).

Lisp Mode

Lisp Mode sets things up for editing Lisp code. It puts `Indent-For-Lisp` on `TAB`.

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing `Base` and `Syntax` attributes, Zmacs warns that the file "has neither a `Base` nor a `Syntax` attribute" and announces that it will use the defaults, `Base 10` and `Zetalisp`. See the section "Buffer and File Attributes in Zmacs", page 155.

Text Mode

Sets things up for editing English text. It puts `Tab-To-Tab-Stop` on `TAB`.

Major Editing Modes, cont'd.

Note

Zmacs supports Fortran Mode as a part of FORTRAN 77, the separately priced software product. For more information, see the *User's Guide to the FORTRAN 77 Tool Kit*.

Macsyma Mode

Macsyma Mode enters a mode for editing Macsyma code. It modifies the delimiter dispatch tables appropriately for Macsyma syntax, makes comment delimiters `/*` and `*/`. It puts Indent-Relative on `TAB`.

Midas Mode

Midas Mode sets things up for editing PDP-10 assembly language code.

Bolio Mode

Bolio Mode sets things up for editing Bolio source files. It is like Text Mode, but also makes `c-m-N`, `c-m-:`, and `c-m-*` insert font characters, and makes word-abbrevs for `znil` and `zt`.

Teco Mode

Teco Mode sets things up for editing TECO. It makes comment delimiters be `!*` and `*!`. It puts Indent-Nested on `TAB`, Forward-Teco-Conditional on `m-'`, and Backward-Teco-Conditional on `m-]`.

Pl1 Mode

Pl1 Mode sets things up for editing PL/1 programs. It makes comment delimiters `/*` and `*/`, and puts Indent-For-Pl1 on `TAB`, Roll-Back-Pl1-Indentation on `c-m-H`, and `Pl1dcl` on `c-≡`. Underscore is made alphabetic for word commands.

Electric Pl1 Mode

Electric Pl1 Mode sets things up for editing PL/1 programs. It does everything Pl1 Mode does: it makes comment delimiters `/*` and `*/`, puts Indent-for-Pl1 on `TAB`, Roll-Back-Pl1-Indentation on `c-m-H`, , and `Pl1dcl` on `c-≡`. Underscore is made alphabetic for word commands. In addition, `;` is Pl1-Electric-Semicolon, `:` is Pl1-Electric-Colon, `#` is Rubout, `@` is Clear, `\` is Quoted Insert.

10. Zmacs Speller

Using the Zmacs Speller

The Zmacs Speller is a small, simple set of tools to help you spell words right. The Speller can examine a single word, a Zmacs region or buffer, a file, or a group of files, for spelling errors. When it encounters a word not listed in one of its dictionary files, it alerts you with a message and a menu of possible responses. A large dictionary of common words is included as part of the Zmacs editor. You can add other dictionaries of special spellings, as used by individuals or groups of users on your system, to the list.

The quickest way to see what the Speller can do is to go to a Zmacs text buffer, type an incorrectly spelled or nonexistent word, and then press `m- $\$$` . Say you type "gorax". When you press `m- $\$$` , you see in the minibuffer at the bottom of the screen:

```
"gorax" is unknown and possibly misspelled.
```

A menu appears offering you a choice of **Prompt** and **Accept** and the word "borax", which is the only word in the basic dictionary close to the spelling of "gorax". The menu may also contain the names of one or more dictionaries.

If you meant to type "borax", click on the word on the menu and the spelling is changed.

If you want to accept the spelling "gorax", click on **Accept** or move the mouse cursor off the menu.

If you want some entirely different word, click on **Prompt** and you are prompted to type another word. If that word is spelled correctly, the message

```
Correcting "gorax" to "monkey".
```

appears in the minibuffer. If the word is spelled incorrectly, the change is made anyway, and the following messages

```
Warning: "shimbax" is not in the dictionary.
```

```
Correcting "gorax" to "shimbax".
```

appear in the minibuffer.

If you click on a dictionary name, the spelling is added to that dictionary for the rest of the session. You can also add the new words permanently to a dictionary by saving the dictionary to a file. See the section "`m-X` Save Spell Dictionary", page 191. See the section "`m-X` Save All Spell Dictionaries", page 192.

Once you've tested the `m- $\$$` command, read a file into the buffer. Enter the command `m-X` Spell Buffer and watch what happens. Chances are there is a misspelled word in the buffer, or at least a

Using the Zmacs Speller, *cont'd.*

word that is not in the dictionary. The menu produced by this command has an additional entry, **Accept once**. Click on **Accept once** if you want the Speller to flag the word the next time it appears. Click on **Accept** if you want the Speller to ignore the word for the duration of the spelling check.

NOTE

To the Speller, any word not in the dictionaries is misspelled by definition. Not all words or alternate spellings are in the basic dictionary.

Also, for purposes of the Speller, a word in running text is defined to be a sequence of characters, each of which must be a letter or an apostrophe. The apostrophe is allowed so that contractions and possessives are recognized as words. This definition of a "word" is not exactly the same as that used by the `Zwei` "word" commands (`m-F` and so forth).

The Zmacs Speller Menu

When the Speller encounters a word not in the dictionary in the course of a `m- $\$$` , `m-X` Spell Buffer, Spell Region, or Spell Word command, a warning appears in the minibuffer

"rugnue" is unknown and possibly misspelled.

and a menu gives you a series of actions to choose from.

Prompt

Prompts in the minibuffer for a replacement word. The misspelled word is replaced by the new word. The replacement word is also checked against the dictionaries and you are warned if it isn't found. Any uppercase letters you type are included when the word is replaced. If the questioned word had an initial capital letter, so will the replacement. Any other capital letters in the original word are ignored.

While the menu is showing, pressing `c-P` is the same as clicking on **Prompt**.

Using the Zmacs Speller, *cont'd.*

Accept once

Accepts this spelling just this once. That is, the Speller ignores the spelling this time, but if the same misspelling is encountered again, it challenges the spelling again. Moving the mouse cursor off the menu is the same as clicking on **Accept once**.

While the menu is showing, pressing **c-O** is the same as clicking on **Accept once**.

Accept

Accepts this spelling and considers it a correct spelling for the duration of the command. There is no way to save words permanently using **Accept**. To save words permanently, you must accept them by clicking on a dictionary name.

While the menu is showing, pressing **c-R** is the same as clicking on **Accept**.

DICTIONARY-NAME

Accepts the spelling and adds it to the dictionary for the remainder of the boot session. You can also add the spelling to the dictionary file permanently. To add a dictionary to the list, use the **m-X Read Spell Dictionary** command.

See the section "**m-X Read Spell Dictionary**", page 190.

See the section "**m-X Save Spell Dictionary**", page 191.

See the section "**m-X Save All Spell Dictionaries**", page 192.

suggestion

Whenever possible, the Speller suggests words that are close in spelling to the questioned word. One or more suggestions may appear on

Using the Zmacs Speller, *cont'd.*

the menu. If one of the suggestions is what you intended, click on it and the spelling in the text is changed to the suggested spelling.

Speller Commands for Spelling

The Zmacs Speller allows you to check spelling by word, region, buffer, file, or groups of files (including tag tables).

m- $\$$ (Spell This Word)

m- $\$$

Checks the spelling of the current word. If point is just to the right of a word, that word is checked. This means you can type a word and then press m- $\$$ to check the spelling. If point is on a word, that word is checked.

Use this command to check the spelling of individual words in your current buffer.

m-X Spell Word

m-X Spell Word

Prompts for a word and runs a spelling check on it. If the word is spelled according to the dictionaries, you are informed in the mini-buffer. If the word is not spelled according to the dictionaries, you are informed. If the dictionaries contain any words similar to your misspelling, they are listed also.

Use this command to check the spelling of words before you type them in your buffer.

m-X Spell Region

m-X Spell Region

Runs a spelling check on the current region.

Use this command to check spelling in a region you have marked.

Speller Commands for Spelling, cont'd.

m-X Spell Buffer**m-X Spell Buffer**

Runs a spelling check on the entire current buffer. It does not matter where point is when you issue the command. The entire buffer is checked.

m-X Spell File**m-X Spell File**

Runs a batch-mode spelling check over a file. With wildcards, you can also specify a group of files.

This command prompts for a pathname and also for the name of a buffer where the words not in the dictionaries are to be written. The questioned words are written to a buffer in alphabetical order, but without identification as to the file they came from. You can save this buffer if you wish.

See the section "m-X Tags Spell", page 185.

m-X Tags Spell**m-X Tags Spell**

Runs a batch-mode spelling check on all the buffers of the current tags table and finds all of the words that aren't in any of the dictionaries and writes them (in alphabetical order) into a buffer you specify. You can save this buffer if you wish.

See the section "m-X Spell File", page 185.

Speller Commands for Spelling, *cont'd.*

See the section "Introduction to Tag
Tables and Search Domains", page
122.

Speller Dictionaries

Introduction to Speller Dictionaries

The Zmacs Speller considers a word to be spelled correctly if it is in a dictionary file and to be misspelled if it is not in a dictionary file. The Speller always checks against the basic dictionary provided as part of Zmacs. In addition, the Speller checks against one or more optional site-specific dictionaries and one or more user-specific dictionaries.

Here are some basic definitions of dictionary terms:

- A *dictionary* is an object in the Lisp environment containing a set of words. It has an associated pathname, and a **modified-p** flag. The pathname refers to a dictionary file. The **modified-p** flag indicates whether words have been added to the dictionary since it was read in. If the **modified-p** flag is on, the `m-x Save Spell Dictionary` and `m-x Save All Spell Dictionaries` commands saves the dictionary. When a dictionary is saved, the **modified-p** flag is turned off.
- A *dictionary file* is a file that holds a set of words. It can be a binary (compiled) dictionary or a character (text) dictionary.
 - A binary dictionary is compiled and has the canonical file type "dict" ("dc" on Unix, "dct" on VMS). The binary dictionary is fast to load or dump.
 - A character dictionary can be any file with text in it. There is no naming restriction, but it is clearer to use the file type "text". The words in the file are the words of the dictionary set, minus duplicates. This format can be easily created, examined and modified by editing.
- There is a special *list of dictionaries* used by the `Zwei` commands. This is the dictionaries currently being used. Order in the list doesn't matter. Some dictionaries on the list appear in the correction menu, and some do not. Whether or not to appear on the menu is a property of a dictionary.

Speller Dictionary Management

Every system includes a basic dictionary, which is in the file `SYS:SPELL;BASIC.DICT`. This is a dictionary of about 30,000 common English words. Not all words or alternate spellings are in the basic dictionary. The dictionary does not include the names of Lisp language forms.

You can add dictionaries for yourself as an individual user, or for your site. Any text file can be used as a dictionary. Dictionaries

Speller Dictionaries, *cont'd.*

can be in either binary (compiled) format, or character (text) format.

Adding User-specific Speller Dictionaries

The Zmacs Speller checks by default for a Speller dictionary in your home directory. This is a standard dictionary and is always included on the list of dictionaries if it exists.

The user-specific dictionary is sought in your directory, on your file server, under your name, with the file name "spell" and the file type "dict" or "text".

If you wish to add words to a dictionary as you go along, using the Speller, you must read the dictionary in ahead of time. Then, any time you click on the dictionary name, the word is added to the dictionary in memory. If you wish to add the words to the dictionary file on disk, you can save the dictionary.

See the section "`m-x` Read Spell Dictionary", page 190.

See the section "`m-x` Save Spell Dictionary", page 191.

See the section "`m-x` Save All Spell Dictionaries", page 192.

A dictionary file is just a file of text. Put all the words you want in your dictionary into a file in your top-level directory and you have a user-specific dictionary.

To speed loading, you can compile the file. It is good practice to use the "text" file type for character (text) dictionary files and the "dict" file type for binary (compiled) dictionary files. When loading standard dictionaries, the Speller looks first in your directory for the file named `spell.dict`. If that file is not found, the Speller looks for `spell.text`. See the section "`m-x` Compile Spell Dictionary", page 190.

If you wish to use other dictionaries besides the basic dictionary, you can add them to the dictionary list interactively with the `m-x` Read Spell Dictionary command. See the section "`m-x` Read Spell Dictionary", page 190. You can also include other dictionaries on your dictionary list using `zwei:read-spell-dictionary`. Speller dictionary functions are most effectively used in a `lisp-init` file to do automatic dictionary operations when you log in. See the function `zwei:read-spell-dictionary`, page 193.

For example, evaluating the following form:

Speller Dictionaries, *cont'd.*

```
(zwei:read-spell-dictionary "f:>skimpy>my-old-list.foo")
```

reads that file into memory as a dictionary and adds *MY-OLD-LIST* to the dictionaries named on the menu. In most cases, the menu lists just the name of the file and not the rest of the pathname, but if two dictionary files have the same name, then the menu lists the pathname for both (name-first, like Zmacs buffer names).

Conversely, evaluating this form:

```
(zwei:read-spell-dictionary "f:>skimpy>my-old-list.foo" nil)
```

reads the file into memory as a dictionary, but does not add it to the dictionaries named on the menu. See the function `zwei:read-spell-dictionary`, page 193.

Note: If you use `zwei:read-spell-dictionary` in your `lispm-init` file, you must also use `zwei:read-standard-spell-dictionaries` if you want to use the standard dictionaries. `zwei:read-spell-dictionary` overrides the auto-loading of the standard dictionaries.

See the section "Adding Site-specific Speller Dictionaries", page 189.

**Adding Site-specific
Speller Dictionaries**

The Zmacs Speller loads the basic dictionary in `SYS:SPELL;BASIC.DICT` when you first use the speller. It is not loaded until you request it or need it.

You can also have other dictionaries designated as site-specific. Site-specific dictionaries are always included on the dictionary list for each user except for those users who override the auto-loading of standard dictionaries by evaluating `zwei:read-spell-dictionary` without also evaluating `zwei:read-standard-spell-dictionaries`. See the section "Adding User-specific Speller Dictionaries", page 188.

You can make a dictionary site-specific by adding a `USER-PROPERTY` item to the `SITE` object in the namespace with the keyword `SPELL-DICTIONARY` and a value that will be parsed as a pathname. The namespace editor documentation explains how to do this: See the section "Updating the Namespace Database" in *Networks*.

There is a command processor command for creating a Speller dictionary including all the user names for your site.

Speller Dictionaries, *cont'd.*

Create Spell Dictionary From Namespace Command

Create Spell Dictionary From Namespace

namespace pathname

Creates a Speller dictionary with all the user-names, first names, and last names from the list of USER objects in the namespace. (For speed, the command works by accessing the files that hold the namespace database rather than by accessing the actual namespace servers.)

<i>namespace</i>	The namespace you wish the dictionary to represent.
<i>pathname</i>	The pathname of the binary dictionary file to be created.

Once you have created the file, use the namespace editor to make it a site-specific dictionary. See the section "Adding Site-specific Speller Dictionaries", page 189.

Speller Dictionary Commands

Here are the commands for manipulating Speller Dictionaries.

m-X Compile Spell Dictionary

m-X Compile Spell Dictionary

Compiles a character (text) dictionary into binary format for quicker loading. The command prompts for two filenames. The first is the pathname of the character dictionary, which can be any text file. The second is the pathname of the binary dictionary. Binary dictionary files usually have a (canonical) file type of "dict".

m-X Read Spell Dictionary

m-X Read Spell Dictionary Reads a dictionary from a file. By default, the dictionary is added to the

Speller Dictionaries, *cont'd.*

menu. When a dictionary is on the menu you can add words to it as they are checked. With a numeric argument, the dictionary is not added to the menu. You are prompted for the pathname of the dictionary, which can be either character (text) or binary (compiled).

m-X Show Spell Dictionaries

m-X Show Spell Dictionaries

Shows the list of dictionaries currently being used by the Zmacs Speller.

m-X Save Spell Dictionary

m-X Save Spell Dictionary Saves an updated copy of a dictionary with all words added in the current session. You can save any dictionary from the list of dictionaries, but it is only meaningful to save a dictionary from the displayed list, since those are the only dictionaries modified. Pathnames are unchanged. Saving an unmodified dictionary does nothing.

Use **m-X Show Spell Dictionaries** to see the list of dictionaries. Completion of dictionary names is available.

See the section "**m-X Save All Spell Dictionaries**", page 192.

Speller Dictionaries, *cont'd.*

m-X Save All Spell Dictionaries

m-X Save All Spell Dictionaries

Saves updated copies of all modified dictionaries on the list of dictionaries. Dictionary pathnames are unchanged.

Use m-X Show Spell Dictionaries to see the list of dictionaries.

See the section "m-X Save Spell Dictionary", page 191.

m-X Kill Spell Dictionary

m-X Kill Spell Dictionary

Removes a dictionary from the list of dictionaries used by the Zmacs Speller. This command has no effect on the file on disk. Type in the pathname of the dictionary in name-first order (like Zmacs buffer names). Completion of dictionary names is available.

m-X Add Word to Spell Dictionary

m-X Add Word To Spell Dictionary

Adds a word to a dictionary. The command prompts for both the word and the name of the dictionary. The word remains in the dictionary as long as the dictionary remains on the list.

You can also add a word at the time the Speller challenges it, by clicking on the dictionary name.

See the function `zwei:add-words-to-spell-dictionary`, page 194.

Speller Dictionaries, *cont'd.*

m-X Delete Word From Spell Dictionary

m-X Delete Word From Spell Dictionary

Deletes a word from a dictionary. This command prompts for both the word and the dictionary.

See the function `zwei:delete-words-from-spell-dictionary`, page 195.

m-X Show Contents of Spell Dictionary

m-X Show Contents Of Spell Dictionary

Shows all the words in a dictionary. Completion of dictionary names is available. With a numeric argument, the command prompts for a pathname and writes the words to that file. In this way, you can get a text file of a binary dictionary.

Use `m-X Show Spell Dictionaries` to see the list of dictionaries.

Speller Dictionary Functions

<code>zwei:read-spell-dictionary</code> <i>pathname</i> &optional (<i>menu-p</i> <i>t</i>)	<i>Function</i>
---	-----------------

This function is intended primarily for use in `lisp-init` files. It reads a dictionary from a file into virtual memory and adds it to the list of dictionaries used by the Zmacs Speller.

If the optional *menu-p* argument is `nil`, the dictionary is not added to the dictionaries shown on the menu. The default is to show the dictionary on the menu.

`zwei:read-spell-dictionary` overrides the auto-loading of standard dictionaries. If you use this function in your `lisp-init` file, you must also use `zwei:read-standard-spell-dictionaries` if you want the standard dictionaries loaded. See the function `zwei:read-standard-spell-dictionaries`, page 194.

Speller Dictionaries, *cont'd.*

See the section "Speller Dictionary Management", page 187.

zwei:read-standard-spell-dictionaries &key *Function*
for-general-use

This function reads the basic dictionary, any site-specific dictionaries, and, optionally, a user-specific dictionary into memory and makes them a part of the list of dictionaries used by the Zmacs Speller.

If the optional keyword argument *for-general-use* is **nil**, only the basic dictionary and any site-specific dictionaries are read in. The default is to read in those dictionaries plus the user's own user-specific dictionary. Only the user-specific dictionary appears on the menu.

You do not need to evaluate this function in your `lisp-init` file unless you are using `zwei:read-spell-dictionary`, which overrides the auto-loading of standard dictionaries, or unless you wish to set *for-general-use* to **nil**.

See the function `zwei:read-spell-dictionary`, page 193.

See the section "Speller Dictionary Management", page 187.

zwei:add-words-to-spell-dictionary *pathname* *Function*
list-of-words &optional
ok-if-dictionary-not-found

This function adds words to a dictionary in virtual memory. Use it to patch a dictionary in a world load that already has the dictionary loaded. If the optional argument *okay-if-dictionary-not-found* is **t** and no dictionary is found, nothing happens. If the argument is **nil**, which is the default, the function signals an error.

For example,

```
(zwei:add-words-to-spell-dictionary "shoebox:>fred>arfnarf.dict"
  ("roadhog" "birdbrain") t)
```

See the function `zwei:delete-words-from-spell-dictionary`, page 195.

The dictionary file on disk is not affected. To add a word to a dictionary file:

Speller Dictionaries, *cont'd.*

See the section "m-X Add Word to Spell Dictionary", page 192.

See the section "m-X Save Spell Dictionary", page 191.

See the section "m-X Save All Spell Dictionaries", page 192.

zwei:delete-words-from-spell-dictionary *pathname* *Function*
list-of-words &optional
ok-if-dictionary-not-found

This function deletes words from a dictionary in virtual memory. Use it to patch a dictionary in a world load that already has the dictionary loaded. If the optional argument *okay-if-dictionary-not-found* is **t** and no dictionary is found, nothing happens. If the argument is **nil**, which is the default, the function signals an error.

For example,

```
(zwei:delete-words-from-spell-dictionary "bogue:>lorna>snage.dict"  
  ("wonderissimo" "megastupidity") t)
```

See the function **zwei:add-words-to-spell-dictionary**, page 194.

The dictionary file on disk is not affected. To delete a word from a dictionary file:

See the section "m-X Delete Word From Spell Dictionary", page 193.

See the section "m-X Save Spell Dictionary", page 191.

See the section "m-X Save All Spell Dictionaries", page 192.

11. Word Abbreviations

Using Word Abbreviations

Using word abbreviation, you can type short abbreviations in an editing buffer which are expanded into text blocks of any length. Thus, you can substitute short, easily typed made-up words and have commonly used words, phrases, paragraphs, or program elements appear in their place.

Word Abbrev Mode is a Zmacs minor mode. Turn it on with the `m-X Word Abbrev Mode` command. Turn it off with another `m-X Word Abbrev Mode` command. When Word Abbrev Mode is on, you will see the word `Abbrev` in the mode line.

Here is an example of using Word Abbrev Mode. Go to an editing buffer. For this example, make it a text buffer. Enter the `m-X Word Abbrev Mode` command. Type a word into the buffer, such as "Tex". Press `c-X c-A`. You are prompted in the minibuffer `Text mode abbrev for "Tex":`. Type "t1" in the minibuffer. Now type "t1" and press the space bar. The word "Tex" appears in the buffer.

From now on in this session, any time you type "t1", "Tex" will appear. If you want to have "t1" appear in the buffer instead of "Tex", type `c-X U` to Undo the last word abbreviation.

The expansion is triggered by a space, RETURN, or any punctuation mark, including asterisks, ampersands, and other symbols on the top row of the keyboard. The trigger is also inserted into the buffer.

If you want to have an abbreviation expand to more than one word, you can either select a region before typing `c-X c-A` or use a numerical argument with `c-X` for the number of words you want to include in the abbreviation.

`c-X c-A` makes word abbreviations only for the current Zmacs major mode. Using `c-X c-A` in a Lisp buffer makes abbreviations that work in all Lisp mode buffer. `c-X c-A` in a Text mode buffer makes abbreviations that work in all Text Mode buffers, and so forth. These are called *mode word abbrevs*. You can make *global word abbrevs* with `c-X +`. Global word abbrevs work in any buffer except those where a mode word abbrev is defined using the same abbreviation. That is, you could have "t1" with a global definition of "(FUNCALL STREAM :STRING-OUT STRING)" and still have "t1" expand to "Tex" in text buffers.

If you want to see a list of word abbreviations, use the `m-X List Word Abbrevs` command.

If you want to save a file of word abbreviations, use the `m-X Write Word Abbrev File` command. To use the word abbreviations in a file, use the `m-X Read Word Abbrev File` command.

Using Word Abbreviations, *cont'd.*

You can edit the current word abbreviations with the `M-X Edit Word Abbrevs` command. You can put all your word abbreviations in a buffer with the `M-X Insert Word Abbrevs` command and then write out the buffer so you can have all your word abbreviations in a readable form. Word abbreviation files are written in a special format with the "qwabl" file type and are not readable.

To stop using word abbreviations, you can either turn the mode off with the `M-X Word Abbrev Mode` command, or use the `M-X Kill All Word Abbrevs` command, which eliminates all defined word abbreviations.

You have some control over capitalization in Word Abbrev Mode. For instance, with "wabv" as an abbreviation for "word abbreviation", "Wabv" expands to "Word abbreviation" and "WABV" expands to "WORD ABBREVIATION". Capitalization is also controlled by how you enter the word abbreviation in the first place.

Word abbreviations work in the minibuffer, so you can even abbreviate commands. If you always type "otehr" when you mean "other", then make "otehr" the abbreviation for "other". You can also use word abbreviation in programming, for writing reports, or in many other editing contexts.

Word Abbreviation Commands

The word abbreviation commands in this section are listed in alphabetical order.

c-X plus-SIGN Add Global Word Abbrev

c-X Plus-sign

Prompts for a global word abbreviation. The default is to make a global abbreviation for the word preceding point. With a numerical argument, the command makes a global abbreviation for that many words before the point. If a region is defined, the command makes a global abbreviation for the region.

Global abbreviations work in all buffers unless a mode word abbreviation is defined for the current buffer mode. See the section "**c-X c-R Add Mode Word Abbrev**", page 200.

c-X c-R Add Mode Word Abbrev

c-X c-R

Prompts for a mode word abbreviation. The default is to make a mode abbreviation for the word preceding point. With a numerical argument, the command makes a mode abbreviation for that many words before the point. If a region is defined, the command makes a mode abbreviation for the region.

Mode abbreviations work in all buffers of the same mode as they were created in. Lisp mode abbreviations work in all Lisp Mode buffers. Text mode abbreviations work in all Text Mode buffers, and so forth. Mode word abbreviations override global word abbreviations for

Word Abbreviation Commands, *cont'd.*

buffers of the same mode. That is, you can have a global abbreviation that works in all buffers except Text Mode, and have the same abbreviation expand differently in Text Mode buffers. See the section "c-X plus-SIGN Add Global Word Abbrev", page 200.

For a prompting form of this command: See the section "Make Word Abbrev", page 202.

Edit Word Abbrevs

Edit Word Abbrevs (m-X) Allows you to edit word abbreviations. Displays word abbreviations in a Word-Abbrev buffer that you can edit in the usual fashion.

Insert Word Abbrevs

Insert Word Abbrevs (m-X) Inserts a list of word abbreviations and their expansions into the buffer. You can make a file of this buffer so you will have a readable list of the word abbreviations you are using. word abbreviation files are not readable.

Kill All Word Abbrevs

Kill All Word Abbrevs (m-X) Eliminates all word abbreviations, whether read in from a file or created interactively with c-X c-A.

Word Abbreviation Commands, *cont'd.*

List Some Word Abbrevs

List Some Word Abbrevs (m-X)

Lists word abbreviations or expansions that contain a given string. Prompts for the string. If you have "tl" as an abbreviation for "Tex", either "t" or "x" will list the abbreviation.

The command m-X List Word Abbrevs, lists all abbreviations and expansions. See the section "List Word Abbrevs", page 202.

List Word Abbrevs

List Word Abbrevs (m-X) Lists all word abbreviations and expansions.

Make Word Abbrev

Make Word Abbrev (m-X) Prompts for and creates a new mode word abbreviation. Note that this command has the same effect as c-X c-R Add Mode Word Abbrev. It does **not** make a global abbreviation.

See the section "c-X c-R Add Mode Word Abbrev", page 200.

Read Word Abbrev File

Read Word Abbrev File (m-X)

Reads in a word abbreviation file created with m-X Write Word Abbrev File. Abbreviations in the file override previously abbreviations, but you can add new abbreviations interactively.

Word Abbreviation Commands, cont'd.

Word abbreviation files are not in a readable form. They have the file type .qwabl.

c-X U Unexpand Last Word**c-X U Unexpand Last Word**

Undoes the last expansion of a word abbreviation, leaving the unexpanded abbreviation. Thus, if you have "t1" as an abbreviation for "Tex", but should, for some reason, want to have "t1" in your text, you could type "t1" and then type c-X U, to get rid of "Tex" and leave "t1" behind.

Word Abbrev Mode

Word Abbrev Mode (m-X) Turns on Word Abbrev Mode. If Word Abbrev Mode is already on, this command turns it off. Word Abbrev Mode is a Zmacs Minor Mode. The word Abbrev appears in the mode line when Word Abbrev Mode is on.

Write Word Abbrev File**Write Word Abbrev File (m-X)**

Takes all the current word abbreviations and puts them in a word abbreviation file. Word abbreviation files are not in a readable form. They have the file type .qwabl.

Use the m-X Read Word Abbrev File to read in a previously created file of abbreviations. See the section "Read Word Abbrev File", page 202.

12. Using Character Styles in Zmacs

Introduction to the Character Style Commands

A number of Zmacs commands allow you to use different character styles. Using character styles, you can indicate program structure with different character styles, or you can do certain kinds of text formatting.

For another kind of text formatting: See the section "Zmacs Commands for Formatting Text", page 35.

The information in this chapter deals with these commands only. For more information on character styles: See the section "Character Styles" in *Converting to Genera 7.0*. For more information on using character styles in programs: See the section "Overview of Character Environment Facilities" in *Programming the User Interface, Volume A*.

A number of choices of character styles are available on the Genera system. Here are a few examples:

Dutch.Bold-Italic.Small

Swiss.Bold-Condensed-Caps.Normal

Eurex.Italic.Huge

Dutch.Roman.Normal

Zmacs commands allow you to specify the character style for a character, word, region, or buffer. (Not all character styles work equally well on all printers.)

Character styles are identified by three characteristics that affect how the character appears. These are family, face, and size. The names of character styles incorporate these three characteristics. Thus, as in the example, you see the family *Eurex*, the face *Italic*, and the size *Huge*; this is expressed in commands as *EUREX.ITALIC.HUGE*. This is a *fully-specified* character style.

Where you see a character style named as *NIL.NIL.NIL* (with *nils* in its name), this indicates that the character style is being merged against a default style. For instance, the default character style for Zmacs buffers is *FIX.ROMAN.NORMAL*. Thus, if you wish a character in a Zmacs buffer to be **bold**, it can be entered as *NIL.BOLD.NIL*. This means that in the Zmacs buffer, its style is merged against *FIX.ROMAN.NORMAL* to produce *FIX.BOLD.NORMAL*. In some other context,

Introduction to the Character Style Commands, *cont'd.*

NIL.BOLD.NIL might be interpreted as SWISS.BOLD.NORMAL because it was merged against SWISS.ROMAN.NORMAL. A character style specification in the form NIL.BOLD.NIL is called a *character face* specification (in contrast to the fully qualified specification).

Here is an example of using character styles. In a Zmacs buffer, type a word. Move the cursor back to the beginning of the word. Press `m-J`. You are prompted

Change character style of word to [default nil.nil.nil]:

Type in NIL.BOLD.NIL and press RETURN. The word you have selected appears in **bold**.

Now move to the beginning of another word. Press `m-J`. You see the same prompt. This time move the mouse over the word you just made bold. You'll see **Mouse-L: NIL.BOLD.NIL** in the mouse documentation line. Click (L). The word you selected is **emboldened**.

Now move to the beginning of another word. Press `m-J`. You see the same prompt. The arrow cursor points straight up and you'll see **Mouse-R: Character style menu** in the mouse documentation line. (If you don't see it, move the mouse a bit and it will show up.) Click (R) and a menu appears.

```

Character Face Code
---Unspecific (family NIL)---
Roman
  Italic
  Bold
  Bold Italic
---Code (family Fix)---
Roman
  Italic
  Bold
  Bold Italic
---Text (family Swiss)---
Roman
  Italic
  Bold
  Bold Italic
Other style, by Family/Face/Size ...
  
```

Move the mouse over the first word **bold** and click any button. The word you selected appears in **bold**.

The menu allows you many choices of character styles. If you click on the item

Other style by Family/Face/Size ...

Introduction to the Character Style Commands, *cont'd.*

you get a series of menus that allows you to select the family, faces within that family, and styles available for that combination of family and face. Not all families have all faces and sizes available. Sizes are relative, for example, SMALL or SMALLER, not absolute.

These are the three ways of selecting a character style. All commands for changing character styles allow you to use these three methods of specifying the character style you want.

A common desire is to include a word or phrase in the same family, but **bold** or *italic* face. If you are typing in a normal Zmacs buffer (FIX.ROMAN.NORMAL) you can make a word bold by pressing `m-J` and then simply entering "bold" or just "b" in response to the prompt. This is specifying a character face.

There are six commands for changing character styles.

`c-J` (Change Style Character) changes the style of a character, or several characters if you use a numeric argument.

`m-J` (Change Style Word) changes the style of a word, or several words if you use a numeric argument.

`c-X c-J` (Change Style Region) changes the style of a region.

`m-X` Change One Style Region changes one style in a region, but not any other. Thus, if you had a region with both **bold** and *italic* in it, you could change the *italic* characters without affecting the **bold** ones.

`m-X` Set Default Character Style sets the default character style for the whole buffer. The command also prompts to ask if you want to set the default character style in the attribute list as well.

`c-m-J` (Change Typein Style) changes the character style for newly inserted characters.

There are two commands for getting information about character styles in a buffer.

`m-X` Show Character Styles displays all character styles in the buffer or in the region, if there is one. The display includes the character style, that is, what you type in to select a character style; the Lisp name of the character style; the font equivalent;

Introduction to the Character Style Commands, *cont'd.*

and samples of the character style. When samples are displayed, you can click on the samples to select that character style for any of the commands for changing character styles.

`m-x` Find Character in Style searches forward for the next character in a given style. You supply the name of the character style as in the commands for changing or specifying character styles.

Character Style Commands in Zmacs

The character style commands in this section are listed in alphabetical order.

Change One Style Region

Change One Style Region (m-X)

Allows you to change one character style in a region without affecting other character styles in the region. This command prompts you for the style you want to change and the style to which you want it changed. You can identify the old and new styles by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

To change all the characters in a region to a single style: See the section "c-X c-J Change Style Region", page 210.

c-J Change Style Character

c-J Change Style Character

This command changes the character style of a single character, or, with a numeric argument, more than one character. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

c-X c-J Change Style Region

c-X c-J Change Style Region

This command allows you to change the character style of a region to a

Character Style Commands in Zmacs, *cont'd.*

single character style. That is, if there are two character styles in the region, both will be changed to the character style you choose with this command. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

To change only one character style in a region: See the section "Change One Style Region", page 210.

m-J Change Style Word

m-J Change Style Word This command changes the character style of a single word, or, with a numeric argument, more than one word. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

c-m-J Change Typein Style

c-m-J Change Typein Style This command sets the character style for newly inserted characters. It does not affect the default style for the buffer. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Character Style Commands in Zmacs, *cont'd.*

Find Character In Style

Find Character in Style (m-X)

This command searches forward for the next character in a given style. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Set Default Character Style

Set Default Character Style (m-X)

This command sets or changes the character style associated with the buffer. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Show Character Styles

Show Character Styles (m-X)

This command displays all the character styles in a buffer. The display includes the character style, that is, what you type in to select a character style; the Lisp name of the character style; the font equivalent; and samples of the character style. When samples are displayed, you can click on the samples to select that character style for any of the commands for changing character styles.

13. Changing Case and Indentation in Zmacs

Changing Case

Overview

Zmacs offers extended commands that convert the case of the code for words, regions, and buffers.

Changing Case of Words

`m-C` Uppercase Initial

Puts next word in lowercase, but capitalizes initial character. With an argument, it capitalizes that many words.

`m-L` Lowercase Word

Puts next word in lowercase. With an argument, it puts that many words in lowercase.

`m-U` Uppercase Word

Puts next word in uppercase. With an argument, it puts that many words in uppercase.

Changing Case of Regions

`c-X c-U` Uppercase Region

Uppercases the region.

`c-X c-L` Lowercase Region

Lowercases the region.

Uppercase Code in Region (`m-X`)

Converts all code (not comments, strings, or quoted characters) to uppercase. This gives the same effect as retyping that text while in Electric Shift Lock Mode. It operates on the region if there is one, otherwise it operates on the current definition.

Lowercase Code in Region (`m-X`)

Converts all code (not comments, strings, or quoted characters) to lowercase. It operates on the region if there is one, otherwise it operates on the current definition.

Changing Case of Buffers

Uppercase Code in Buffer (`m-X`)

Converts all code (not comments, strings, or quoted characters) to uppercase. This gives the same effect as retyping that text while in Electric Shift Lock Mode. It queries for a buffer name (the default is the current buffer) and operates on that buffer.

Changing Case, *cont'd.*

Lowercase Code in Buffer (n-X)

Converts all code (not comments, strings, or quoted characters) to lowercase. It queries for a buffer name (the default is the current buffer) and operates on that buffer.

Indentation

Overview

Proper indentation helps make complicated Lisp programs readable. Indentation should reflect the structure of a program. An expression should be indented so that its subforms are easily identifiable, and so that a function can be related to its arguments by eye, without counting parentheses.

The indentation commands work in any Zmacs major mode; the TAB key indents differently depending on the mode. When you give an indent command an argument of n , n equals the number of Space characters in the default font.

Indenting Current Line

TAB

In Lisp mode, the TAB key indents the current line of Lisp code correctly with respect to the line above it. (In most other modes, TAB inserts a Tab character.) Point remains fixed with respect to the code.

With a numeric argument n , it indents the next n lines including the current one, and leaves point at the $n+1$ st line.

c-TAB

Indent Differently

Tries to indent this line differently. If called repeatedly, it makes multiple attempts.

m-TAB

Insert Tab

Inserts a Tab character, even in Lisp Mode, in the buffer at point.

c-m-TAB

Indent For Lisp

Indents this line to make ground (indented) LISP code, even in a mode other than Lisp Mode. A numeric argument specifies the number of lines to indent.

Indentation in zl:loop Macros

The zl:loop Indentor

Zwei now indents code within a **zl:loop** macro in a more attractive way than it did in the past. The TAB key indents the code while recognizing and dealing appropriately with **zl:loop** keyword clauses. This new indentation style is a change in the

Indentation, *cont'd.*

Zmacs user interface for writing Lisp code. You might want to know how to turn it off because it indents new code in a style that is inconsistent with existing code.

To turn off the new `zl:loop` indenter, include the following flag in your init file:

```
(SETF ZWEI:*INHIBIT-FANCY-LOOP-INDENTATION* T)
```

The initial value for this flag is `nil`; `t` reverts to the old-style indenter.

How to Use the zl:loop Indenter

Use the `zl:loop` indenter the same way as always: type a token on a line of code inside a `zl:loop` and then press `TAB`, which correctly indents the code.

The usual sequence:

```
LINE finally TAB
```

(substitute any other `zl:loop` word for `finally`) reindents based on the new knowledge that this is a `finally` line rather than a body line. The `zl:loop` indenter always ignores comments.

Loop Indenter Example 1

The right indentation sometimes depends on forms after the line you are indenting. For example:

```
(loop for a being the array-elements of b
      ;; comment
      do (frob a))
```

Press `TAB` at the end of the comment line and:

```
(loop for a being the array-elements of b
      ;; comment
      do (frob a))
```

happens because the `zl:loop` indenter anticipates that you might instead be doing this:

```
(loop for a being the array-elements of b
      ;; comment
      using (sequence b) (index i)
      do (frob a))
```

Indentation, cont'd.

Loop Indenter Example 2

The **zl:loop** indenter second guesses on a few things, but gets them right after you type a token on a line and press **TAB**. For example:

```
(loop when x
      do (y)
      (z))
```

is indented correctly; this is how the indentation initially reads. If (z) had been do, it would have put the do where the (z) is:

```
(loop when x
      do (y)
      do (z))
```

But pressing **TAB** reindents it correctly:

```
(loop when x
      do (y)
      do (z))
```

The converse can come up, for example:

```
(loop with x
do (z))
```

is fixed with **TAB**:

```
(loop with x
      do (z))
```

```
(loop with x
      = (z))
```

is indented incorrectly until you press **TAB**, resulting in:

```
(loop with x
      = (z))
```

Indentation, *cont'd.*

Centering the Current Line

`m-S` Center Line

Centers the text of the current line within the line. With an argument *n*, it centers *n* lines and moves past them.

Indenting New Line

The keystroke combination RETURN TAB gets you into the right position to start typing the next line of code. LINE is the abbreviation for that combination.

LINE Indent New Line

If the next two lines are blank, goes to the next line; otherwise, it creates a new blank line following the current one. In any case, it does a TAB on that blank line.

Reindenting Expression

`c-m-Q` Indent Sexp

Corrects the indentation of the expression following point by adjusting the amount of space before each line in the expression. `c-m-Q` positions point in front of the incorrectly indented expression. This does not affect the indentation of the current line, but only fixes the indentation of following lines with respect to the current line. Use after modifying an expression.

With a numeric argument of *n*, it fixes the indentation of the next *n* expressions.

Indenting Region

`c-m-\` Indent Region

Indents each line in the region. With no argument, it calls the current Tab command to indent. With an argument of *n*, it indents each line *n* spaces in the current font.

Going Back to First Indented Character

`m-M` Back To Indentation

`c-m-M`
`m-RETURN`
`c-m-RETURN`

Positions point before the first nonblank character on the current line.

Indentation, *cont'd.*

Indenting Region Uniformly

c-X TAB

Indent Rigidly

c-X c-I

Shifts text in the region sideways as a unit. All lines in the region have their indentation increased by the numeric argument of the command (the argument can be negative).

Aligning Indentation

Indent Under (c-n-X)

Fixes indentation to align under *string*, which you click on with the mouse cursor or which you specify in the minibuffer.

When you use the mouse to specify the alignment string, begin by putting the cursor on the line you want to indent, then click right, click on Indent Under, then either point the cursor (a down-arrow pointing at a box) at a character that you want to line up with and click left, or type in a string for which it searches.

When you type the alignment string in the minibuffer, it searches back, line by line, forward in each line, for a string that matches the one read and that is farther to the right than the cursor already is. It indents to align with the string found, removing any previous indentation first.

Deleting Indentation

m-^

Delete Indentation

c-m-^

Deletes the newline character and any indentation at the beginning of the current line. It tacks the current line onto the end of the previous line, leaving one space between them when appropriate, for example, at the beginning of a sentence.

With any numeric argument, it moves down a line first, thus killing the end of the current line.

New Line with This Indentation

m-O

This Indentation

Makes a new line after the current one, deducing the new line's indentation from point's position on the current line. If point is to the left of the first nonblank character on the current line, it

Indentation, *cont'd.*

indents the new line exactly like the current one. But if point is to the right of the first nonblank character, it indents the new line to the current position of point. Regardless, it leaves point at the end of the newly created line.

With a numeric argument, the new line is always indented like the current one, no matter where point is. With an argument of zero, it indents current line to point.

Moving Rest of Line Down

c-n-0

Split Line

Moves rest of current line down one line. It inserts a carriage return and indents new line directly beneath point. With a numeric argument n , it moves down n lines.

Inserting Blank Line

c-0

Make Room

Inserts a blank line after point. With a numeric argument n , it inserts n blank lines.

Deleting Blank Line

c-X c-0

Delete Blank Lines

Deletes any blank lines around the end of the current line.

14. Editing Lisp Programs in Zmacs

Introduction

Lisp Machine programmers develop programs in repeated cycles, each a sequence of editing, compiling, testing, and debugging. These cycles are often nested. Zmacs allows you to edit and test large programs dynamically, without frequent file system operations. This manual does not describe any style of interacting with the environment in developing Lisp programs. See the section "Program Development Tools and Techniques" in *Program Development Utilities*. It focuses on the interaction between programmers and the Lisp Machine, presenting ways of using helpful Lisp Machine features and tools during each stage of program development.

As a programmer on a Lisp Machine you typically read a file containing Lisp code into an editor buffer, make modifications, test the results, make more changes, and so on, until satisfied with the behavior of the program. Only then do you need to write the buffer back out to the file system. The debugging loop is much tighter and more responsive than in traditional programming environments. You can even evaluate Lisp forms directly from inside the editor, without returning to a Lisp Listener. Alternatively, you can divide the screen into a Lisp Listener window and a Zmacs window, so that you can direct your attention to either without changing the display.

Zmacs provides extensive features for locating source code of specified functions. If an error occurs, the Debugger can cause Zmacs to read in the source of the function that got the error. You can then debug and recompile the function. Similar features complement the message-passing capabilities of the Zetalisp language.

When you edit a file with a Lisp type, Zmacs puts that buffer into Lisp mode. A command exists for explicitly placing a buffer in Lisp mode:

Lisp Mode (M-X)

Lisp Mode

Places the current buffer into Lisp mode.

Base and Syntax Default Settings for Lisp

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Zetalisp. See the section "Buffer and File Attributes in Zmacs", page 155.

Commenting Lisp Code, *cont'd.*

comments on the current line if no region exists. This command can be reversed with Undo if no other undoable command has intervened.

Moving Down to Comment on Next Line

`m-N`

Down Comment Line

Moves point to the beginning of the comment on the next line. If there is no comment on the next line, it creates one. If the comment on the current line is empty, it deletes it before going to the next line.

With a numeric argument n , it moves point to the beginning of the comment on the n th line after the current one.

Moving up to Comment on Previous Line

`m-P`

Up Comment Line

Moves point to the beginning of the comment on the previous line. If there is no comment on the previous line, it creates one. If the comment on the current line is empty, it deletes it before going on to the previous line.

With a numeric argument n , it moves point to the beginning of the comment on the n th line before the current one.

Setting the Comment Column

`c-X ;`

Set Comment Column

Sets the comment column to be the current horizontal position of the cursor.

With a numeric argument, it finds the nearest comment above the current line, sets the comment column to line up with that comment, and actually puts a comment on the current line at that column.

Creating a New Indented Comment Line

`m-LINE`

Indent New Comment Line

Makes a new blank line after the current line and starts a new comment there, indented properly. If there was already a

Commenting Lisp Code, *cont'd.*

comment on the current line, the comment on the new line is of the same kind. (That is, it has the same number of semicolons and is indented the same.) If there was no comment on the starting line, `m-LINE` starts a new line, indenting the new line as appropriate for the major mode.

Inserting and Removing Lisp Comments From Regions

`c-X c-;`

Comment Out Region

Comments out each line in the region. When the region ends at the beginning of a line, it does not comment out that line. If any part of the line is part of the region, then it does comment out that line.

With a numeric argument (`c-U c-X c-;`) the command restores lines in the region that have been commented out. When any part of the line is part of the region, comments are removed from around that line.

The removal works the same as the commenting out. That is, a single semi-colon (;) in column 1 is removed. Removal of comments stops at a line without a semi-colon in column 1, even if more lines that have been commented out remain in the region. The rest of the region does remain in this case, so that you can resume.

Evaluating and Compiling Lisp Programs

Overview

The commands in this section form a link between the Zmacs editor and the Lisp language. They allow the evaluation and compilation of code from Zmacs buffers. These commands are an important part of the debugging loop.

When a Lisp form is being compiled or evaluated, the editor displays a message that classifies what is being compiled.

It classifies macros as functions (because these go in the function cell of a symbol). For example:

```
Compiling Function SUN
Evaluating Variable MARS
Compiling Flavor STAR
```

Evaluating Lisp Programs

m-ESCAPE

Evaluate Minibuffer

Evaluates expressions from the minibuffer. You enter Lisp expressions in the minibuffer, which are evaluated when you press END. The value of the expression itself appears in the echo area. If the expression displays any output, that appears as a typeout window.

Evaluate Into Buffer (m-X)

Evaluates an expression read from the minibuffer and inserts the result into the buffer. You enter a Lisp expression in the minibuffer, which is evaluated when you press END. The result of evaluating the expression appears in the buffer before point. With a numeric argument, it also inserts any typeout that occurs during the evaluation into the buffer.

Evaluate Buffer (m-X)

Evaluates the entire buffer. The result of evaluating the buffer appears in the minibuffer. With a numeric argument, it evaluates from point to the end of the buffer.

Evaluate Region (m-X)

c-sh-E

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

Evaluating and Compiling Lisp Programs, *cont'd.*

c-m-sh-E

Evaluate Region Verbose

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in a typeout window.

Evaluate Region Hack (m-X)

Evaluates the region, ensuring that any Lisp variables appearing in a **defvar** have their values set. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

Evaluate Changed Definitions (m-X)

Evaluates any definitions that have changed in any of the current buffers. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions (the default evaluates all changed definitions).

Evaluate Changed Definitions of Buffer (m-X)

m-sh-E

Evaluates any definitions that have changed in the current buffer. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions (the default evaluates all changed definitions).

Evaluate And Replace Into Buffer (m-X)

Evaluates the Lisp object following point in the buffer and replaces it with its result.

c-m-Z

Evaluate And Exit

Evaluates the buffer and exits Zmacs. It selects the window from which the last **ed** function or the last debugger **c-E** command was executed.

Compiling Lisp Programs

Compile Buffer (m-X)

Compiles the entire buffer. With a numeric argument, it compiles from point to the end of the buffer. (This is useful for resuming compilation after a prior **Compile Buffer** has failed.)

Evaluating and Compiling Lisp Programs, *cont'd.*

Compile Changed Definitions (m-X)

Compiles any definitions that have changed in any of the current buffers. With a numeric argument, it prompts individually about whether to compile particular changed definitions (the default compiles all changed definitions).

Compile Changed Definitions of Buffer (m-X)

m-sh-C

Compiles any definitions that have changed in the current buffer. With a numeric argument, it prompts individually about whether to compile particular changed definitions. The default is to compile all changed definitions.

Compile Changed Definitions of Tag Table

Compile Changed Definitions of Tag Table (m-X)

Compiles any definitions that have changed in any of the buffers in the current tag table. With a numeric argument, the command prompts individually about whether to compile particular changed definitions. The default is to compile all changed definitions.

Compile File (m-X)

Compiles a file, offering to save it first (if it has an associated buffer that has been modified). It prompts for a file name in the minibuffer, using the file associated with the current buffer as the default. It does not load the file.

Load File (m-X)

Loads a file, possibly saving and compiling it first. It prompts for a file name, taking the default from the current buffer. It checks to see if the file you are compiling corresponds to a buffer and offers to save that buffer if it is modified. If the .bin file is older than the .lisp file, it offers to compile the file first. If the typeout window displays any compiler warnings, Load File asks if you really want to load the file despite the compiler warnings.

m-Z

Compile And Exit

Compiles the buffer and exits Zmacs. It selects the window from which the last ed function or the last debugger c-E command was executed.

Evaluating and Compiling Lisp Programs, *cont'd.*

Lisp Compiler Warnings

Compiler warnings are kept in an internal database that you can inspect and manipulate in various ways with several editor commands.

Compiler Warnings (m-X)

Creates the compiler warnings buffer (called *Compiler-Warnings-1*) if it does not exist, puts all outstanding compiler warnings in that buffer, and switches to that buffer. You can view the compiler warnings by scrolling around and doing text searches through them using Edit Compiler Warnings (m-X).

Edit Compiler Warnings (m-X)

Prompts you with the name of each file mentioned in the database, allowing you to edit the warnings for that file. It then splits the Zmacs frame into two windows: the upper window displays a warning message and the lower one displays the source code whose compilation caused the warning. After you have finished editing each function, c-. gets you to the next warning: the top window scrolls to show the next warning and the bottom window displays the function associated with this warning. Successive c-.s take you through all of the warning messages for all of the files you specified. When you are done, the last c-. puts the frame back into its previous configuration.

Edit File Warnings (m-X)

Asks you for the name of the file whose warnings you want to edit. You can give either the source file or the compiled file. Only warnings for this file are edited. If the database does not have any entries for the file you specify, the command prompts you for the name of a file that contains the warnings, in case you know that the warnings are stored in another file.

Load Compiler Warnings (m-X)

Loads a file containing compiler warning messages into the warnings database. It prompts for the name of a file that contains the printed representation of compiler warnings. It always replaces any warnings already in the database.

Parenthesizing Lisp Expressions

m-(

Make ()

Inserts matching parentheses, leaving point between them. With a numeric argument n , it encloses the next n Lisp expressions in parentheses. When the number of expressions requested cannot be satisfied, it beeps and does nothing. With point on the open parenthesis of a **defun**, an argument of 1 encloses the whole **defun** within a new set of parentheses. Any argument larger than 1 would have no effect. In Text Mode, a word or a phrase within parentheses is treated as a Lisp form.

See also the description of the command m-): See the section "Motion Among Top-Level Expressions", page 77.

Matching parentheses in Zmacs Lisp buffers flash in both directions.

The matching open parenthesis flashes when the cursor is sitting just past a close parenthesis and the matching close parenthesis flashes when the cursor is sitting on an open parenthesis.

Expanding Lisp Expressions

Two editor commands allow you to expand macros: **Macro Expand Expression** and **Macro Expand Expression All**.

`c-sh-M` **Macro Expand Expression**

Reads the Lisp expression following point and expands the form itself but not any of the subforms within it. It displays the result in the typeout window. With a numeric argument, it pretty-prints the result back into the buffer immediately after the expression.

`m-sh-M` **Macro Expand Expression All**

Reads the Lisp expression following point, and expands all macros within it at all levels. It displays the result in the typeout window. With a numeric argument, it pretty-prints the result back into the buffer immediately after the expression. It assumes that every list in the expression is a form, so if car of a list is a symbol with a macro definition, the purported macro invocation is expanded.

Locating Source Code to Edit

Introduction

The functions that make up a program or system can depend on each other in complicated ways. When you are editing one function, you sometimes have to go off and look at another function, and possibly modify that one too.

This section describes the Edit Definition command and other commands that list and/or edit various sets of definitions. In addition, two pairs of List and Edit commands help identify changed code by finding or editing *changed* definitions in buffers. By default, the *changed* commands find changes made since the file was read; use numeric arguments to find definitions that have changed since they were last compiled or saved.

The Zmacs Edit

Definition Commands

Edit Definition (`m-.`) is a powerful command to find and edit function definitions, macro definitions, global variable definitions, and flavor definitions. In general, Zmacs treats as a definition any top-level expression having in functional position a symbol whose name begins `def`.

It is particularly valuable for finding source code, including system code, that is stored in a file other than that associated with the current buffer. It finds multiple definitions when, for example, a symbol is defined as a function, a variable, and another type of object. It maintains a list of these definitions in a support buffer.

Zmacs Command: `m-.`

This command is one of the most valuable tools of the system. When you are developing or debugging programs, you can use `m-.` to find the definition of an ordinary function, generic function, flavor, method, variable, package, or other type of definition. Completion is supported on the definition, if it is already in an editor buffer.

`m-.` prompts for a definition to find. You can enter a large variety of representations, and `m-.` figures out what definition you are seeking. For example, you can enter symbols with or without package prefixes.

You can provide any of the following responses to the `m-.` prompt:

Locating Source Code to Edit, *cont'd.*

<i>symbol</i>	Finds the definition of <i>symbol</i> , which can be an ordinary function or generic function. For generic functions, the defgeneric form is found if one exists; all existing methods are also found. <i>symbol</i> can also be one of: variable, package, defstruct structure, flavor, or other types of definitions.
<i>(generic-function flavor)</i>	Finds the definitions of one method that implements <i>generic-function</i> on instances of <i>flavor</i> and asks if you mean that method. If not, it proceeds to find other methods, including special-purpose methods such as :before , :after , :default , and so on.
<i>(symbol property)</i>	Finds the function named by function spec (:property symbol property). This is a handy abbreviation.
<i>function-spec</i>	Finds the definitions of <i>function-spec</i> . For example, you could enter (flavor:method change-status cell) to find the method of that function spec. Often it is more convenient to enter the list (change-status cell) instead.

When the requested Lisp object has multiple definitions, one of them is displayed. You can then use **c-U m-.** to cycle through the other definitions. Also, a list of all definitions and the files they are located in is stored in a buffer called ***Definitions-n***. The position of the cursor in that buffer controls where **c-U m-.** will go next.

You can also point at forms with the mouse, in a buffer or in other windows, and click **m-left** to edit the definition.

Example of the m-. Command

Suppose you are modifying a function called **sun**, which was written by someone else. **sun** calls the unfamiliar **luna**, and you need to find out what **luna** does before proceeding. Use **m-.** to peek at the definition of **luna**.

When you type **m-.**, Zmacs prompts you for the name of a definition. If point is in the expression where **luna** is called, the default name is **luna**, and you need only press **END**. If point is somewhere else and the default is wrong, you can point at the

Locating Source Code to Edit, *cont'd.*

word **luna** with the mouse or you can type it in. To let you know that you can define a name with the mouse, the mouse cursor changes to an arrow pointing straight up. All the symbols that are names of definitions you could specify become mouse sensitive.

Edit Installed Definition (m-X)

Edits the installed version of the file that contains the definition of a specified Lisp object. It prompts for the name of the definition; if one of your buffers already contains the installed version of that definition, it selects that buffer. Otherwise, it reads in the source file that contains the definition. It always positions the cursor in front of the definition. When the object has more than one definition, use a numeric argument to edit another definition of the same object. You can repeat this until there are no more definitions of that object.

Edit Changed Definitions (m-X)

Determines which definitions in any Lisp Mode buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed since the buffer was read in and selects the first one on the list. Use `c-.` (Next Possibility) to move to subsequent definitions. See the section "Displaying the Next Possibility", page 126.

Edit Changed Definitions accepts a numeric argument to control the time point for determining what has changed:

Value Meaning

- 1 For each buffer, since the file was last read (the default).
- 2 For each buffer, since the buffer was last saved.
- 3 For each definition in each buffer, since the definition was last compiled.

Edit Changed Definitions of Buffer (m-X)

Determines which definitions in the current buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed since the buffer was read in and selects the first one on the list. Use `c-.` (Next Possibility) to move to subsequent definitions. See the section "Displaying the Next Possibility", page 126.

Locating Source Code to Edit, *cont'd.*

Edit Changed Definitions of Buffer accepts a numeric argument to control the time point for determining what has changed:

Value Meaning

- | | |
|---|---|
| 1 | Since the file was last read (the default). |
| 2 | Since the buffer was last saved. |
| 3 | Since the definition was last compiled. |

Edit Cp Command

Edit CP Command (m-X)

Reads the source of a CP command into an editor buffer. With a numeric argument, prompts for a comtab. Otherwise it looks for the command in the global comtab.

Edit System Files

Edit System Files (m-X)

Reads all of the files of a system into buffers. With a numeric argument, the files of the component system are also read into buffers.

The List Definition Commands

List Definitions (m-X)

Displays the definitions in a specified buffer. It reads the buffer name from the minibuffer, using the current buffer as the default. It displays the list as a typeout window. The individual definition names are mouse sensitive.

List Duplicate Definitions (m-X)

Displays the duplicate definitions in the current buffer, if any. This is especially useful for checking patch files or files made by merging several programs together. c-. (Next Possibility) moves point to duplicate definitions that occur earlier in the file, beginning with the earliest duplicate and not including the latest duplicate. See the section "Displaying the Next Possibility", page 126.

List Changed Definitions (m-X)

Displays a list of any definitions that have been edited in any

Locating Source Code to Edit, *cont'd.*

buffer. Use `c-. (Next Possibility)` to start editing the definitions in the list. See the section "Displaying the Next Possibility", page 126.

List Changed Definitions accepts a numeric argument to control the time point for determining what has changed:

Value Meaning

- 1 For each buffer, since the file was last read (the default).
- 2 For each buffer, since the buffer was last saved.
- 3 For each definition in each buffer, since the definition was last compiled.

List Changed Definitions of Buffer (`m-X`)

Displays the names of definitions in the buffer that have changed. It makes an internal list of the definitions changed since the buffer was read in and offers to let you edit them. Use `c-. (Next Possibility)` to move to subsequent definitions. See the section "Displaying the Next Possibility", page 126.

List Changed Definitions of Buffer accepts a numeric argument to control the time point for determining what has changed:

Value Meaning

- 1 Since the file was last read (the default).
- 2 Since the buffer was last saved.
- 3 Since the definition was last compiled.

The Edit Callers Commands

When you are modifying a large system, you often have to make sure that changing a function does not render unusable other functions that call the modified one. Zmacs provides facilities for editing the sources of all the functions defined in the current world that call a given one. This removes some of the unpleasantness of making incompatible changes to large programs and is a good example of how Zmacs interacts with the Lisp environment to make programming easier.

Locating Source Code to Edit, *cont'd.*

Edit Callers (m-X)

Prepares for editing all functions that call the specified one. The prompt is the same kind that Edit Definition gives you. It reads a function name via the mouse or from the minibuffer with completion.

It selects the first caller; use c-. (Next Possibility) to move to a subsequent definition. See the section "Displaying the Next Possibility", page 126.

Multiple Edit Callers (m-X)

Prompts for the names of a group of functions and edits those functions in the current package that call *any* of the specified ones. It reads a function name from the minibuffer, with completion, initially offering a default function name. It continues prompting for more function names until you end the list with RETURN.

Use m-X Multiple Edit Callers Intersection when you want to edit those functions that call *all* of the specified functions. See the section "Multiple Edit Callers Intersection", page 240.

Multiple Edit Callers Intersection

Multiple Edit Callers Intersection (m-X)

Prepares for editing all the functions that call *all* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for a function name until you end it with just a carriage return.

Use m-X Multiple Edit Callers if you want to edit all the functions that call *any* of the specified functions. See the section "Multiple Edit Callers".

List Callers (m-X)

Prompts for the name of a function exactly the way Edit Callers does, but instead of editing the callers in the current package of the specified function, it simply displays their names. The names are mouse-sensitive. If you point at one and click left, you can edit the source of that caller. If you click right, a menu pops up that offers to give the argument list of the selected caller, to disassemble it, to edit it, or to see its documentation string. In addition, c-. (Next Possibility) works in this context, offering the first caller to be edited, and queuing up the other callers to be edited in sequence.

Locating Source Code to Edit, *cont'd.*

Multiple List Callers (m-x)

Lists all the functions that call *any* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for more function names until you end the list with RETURN.

The list of function names is mouse-sensitive: see List Callers (m-x). c-. (Next Possibility) edits the callers. See the section "Displaying the Next Possibility", page 126.

Use m-x Multiple List Callers Intersection when you want only callers that call *all* the specified functions. See the section "Multiple List Callers Intersection", page 241.

Multiple List Callers Intersection

Multiple List Callers Intersection (m-x)

Lists all the functions that call *all* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for a function name until you end it with just a carriage return.

Use m-x Multiple List Callers if you want to list functions that call *any* of the the specified functions.

See the section "The Zmacs Edit Callers Commands", page 239.

Patching

For complete information about patching: See the section "Patch Facility" in *Program Development Utilities*.

Making Patches

During a typical maintenance session you might make several changes to existing definitions or write new ones. Rather than recompiling the entire system every time you change a source file, you can copy only the new or revised code into a *patch file* and write the file ("finish" the patch). Whenever you finish a patch, the patch facility automatically compiles the file and records the event in a "patch registry" for the system, noting the number of the patch, the system being patch, and a brief user-supplied description. As soon as a user loads the patch file (after the system is loaded), the state of the given system in his or her machine is presumably the same as in the developer's machine when the patch was finished.

The patch facility allows you to have several patches in progress at once. Thus you can patch several different systems or several different minor versions of the same system during one work session. The patch facility manages this potentially dangerous situation in the following way. Every time you start a patch, a number and a place in the patch registry is reserved for the patch in production. The patch is marked *in-progress*. When the patch is finished, the entry is completed and the in-progress mark removed. If you decide to abort the patch, the registry entry is automatically deleted.

The ability to have more than patch in-progress to more than one system makes it imperative that you keep track of the state of your various patches. If a patch is left unfinished (unwritten), the `load-patches` function will load neither the in-progress patch or any subsequent finished patches.

The patch facility considers patches to be active or inactive and in one of the following states: initial, in-progress, aborted, or finished. `Show Patches (m-X)` displays the state of all patches started in this work session. If more than one patch is in progress, one of them is known as the *current patch*. The commands that add patches, like `Add Patch (m-X)`, add only to the patch considered by the patch facility to be the current patch. The command `Select Patch (m-X)` displays a menu of active patches and allows you to make another patch the current one.

In general you should adhere to the following steps in making a patch. It is assumed that your system is patchable; that is, the `:patchable` option appears in the system declaration.

Patching, *cont'd.*

1. You must load (via `load-system`) the major version of the system that you want to patch.
2. Read in the source files you want to edit into a Zmacs buffer. Make all changes and test them thoroughly. Write the source file.
3. Use the appropriate Zmacs commands to make your patch. Begin the patch, using `Start Patch (n-X)`.
4. Add the changed code to the patch buffer by using `Add Patch (n-X)`, `Add Patch Changed Definitions of Buffer (n-X)`, or `Add Patch Changed Definitions (n-X)`.
5. Finish the patch, using `Finish Patch (n-X)`, or abort the patch, using `Abort Patch (n-X)`.

Amusingly

Commands provided for initiating a patch are `Start Patch (n-X)`, `Start Private Patch (n-X)`, and `Add Patch (n-X)`.

Start Patch (n-X)

Starts a new patch, prompting you for the name of the system to be patched; it must be a system currently loaded. It assigns a new minor version number for that particular system by writing a new version of the patch directory file with an entry for that minor version number. The patch is marked as in-progress. It starts constructing the patch file in an editor buffer, but does not select the buffer.

While you are making your patch file, the minor version number that has been allocated for you is reserved so that nobody else can use it. Thus, if two people are patching the same system at the same time, they cannot be assigned the same minor version number.

The command does not actually move any definitions into the patch file. You must explicitly do so with `Add Patch Changed Definitions of Buffer (n-X)`, `Add Patch Changed Definitions (n-X)`, or `Add Patch (n-X)`.

The patch facility permits you to start another patch before finishing the current one. However, if your new patch is to the same system, the patch facility warns you that you already have a patch in progress and allows you to take one of four actions:

- Abort the in-progress patch and start a new patch.
- Finish the in-progress patch and start a new patch.
- Proceed with the second patch (initial patch) for this system and leave the in-progress patch intact.

Patching, *cont'd.*

- Use the existing buffer and do not start a new patch.
-

Start Private Patch (m-X)

Although similar to Start Patch (m-X), Start Private Patch (m-X) does not have any relationship to systems, major and minor version numbers, and official patch directories. Rather it allows you to make a private patch file that you can load, test, and share with other users before you install a numbered patch that is automatically available to all users.

Instead of prompting for a system name, the command prompts for a file name. Start Private Patch does not actually move any definitions into the patch file. Use Add Patch Changed Definitions of Buffer (m-X), Add Patch Changed Definitions (m-X), or Add Patch (m-X) to insert the code. Finishing the patch (using Finish Patch (m-X)) writes it out to the specified file.

Note: Use the Load File command or Load File (m-X) to load a private patch; the Load Patches command and the **load-patches** function do not load private patches.

Add Patch (m-X)

Starts a new patch if none is underway, prompts you for a system name, and inserts the region or current definition into the patch buffer. If a patch was in progress, Add Patch (m-X) just adds the region or current definition to the current patch file.

If you mistakenly use the command on code that does not work, select the buffer containing the patch file and delete it. Then later you can use Add Patch (m-X) on the corrected version. For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

Add Patch (m-X), Add Patch Changed Definitions (m-X), or Add Patch Changed Definitions of Buffer (m-X) insert code into the patch file. If the patch is being made to the system the current buffer's file came from, the commands proceed.

If there is a current patch, and it is not appropriate for the system that the buffer's file came from, you see a menu showing all of the current patches, plus an option to create a new patch appropriate for the buffer, plus an option to abort.

Patching, *cont'd.*

Add Patch Changed

Definitions of Buffer (m-X)

Add Patch Changed Definitions of Buffer (m-X) selects each definition that was changed in the buffer and asks you whether or not you want the definition patched.

For each definition, you can respond as follows:

<i>Response</i>	<i>Action</i>
Y	Patches the definition.
N	Skips the definition.
P	Patches the definition and any additional modified definitions in the same buffer without asking any more questions.

A definition needs to be patched if it has been changed since it was last patched or if it has not been patched since the file was read into the buffer.

For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

Add Patch Changed

Definitions (m-X)

Add Patch Changed Definitions (m-X) selects a buffer in which definitions were changed and asks whether or not you want to patch the changed definitions. Answering N skips the buffer and proceeds to the next buffer, if any. Answering Y selects each definition that has changed in that buffer and asks you whether or not you want the definition patched. For each definition, you can respond as follows:

<i>Response</i>	<i>Action</i>
Y	Patches the definition.
N	Skips the definition.
P	Patches the definition and any additional modified definitions in the same buffer without asking any more questions; when done, it proceeds to the next buffer.

If there are more buffers containing definitions to be patched, it asks questions again when it gets to the next buffer.

Patching, *cont'd.*

A definition needs to be patched if it has been changed since it was last patched or if it has not been patched since the file was read into the buffer.

For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

When making multiple patches during one work session use the Select Patch and Show Patches commands to keep track of patches.

Select Patch (m-X)

When you are making more than one patch during a work session, Select Patch (m-X) allows you to choose a different patch as the current patch from a menu of active patches. The patching commands (like Add Patch and Add Patch Changed Definitions of Buffer) insert definitions into the patch file that you have selected as the current patch. To insert patch definitions into another buffer, use Select Patch to choose that buffer as the current patch.

Show Patches (m-X)

Show Patches (m-X) displays the state of all patches started in this session. Patches are either active or inactive and can be in one of the following states: initial, in-progress, aborted, or finished. *Inactive patches* are in an aborted or finished state. *Active patches* are in an initial or in-progress state. *Initial* means that the patch buffer has been initialized but as yet no definitions have been added to the buffer. *In-progress* means that the patch buffer has been initialized and definitions have been added to the buffer.

Show Patches groups the active and inactive patches and identifies the current patch.

After making and testing all of your patches, use the Finish Patch command to install the patch in the system.

Patching, *cont'd.*

Finish Patch (m-X)

Finish Patch (m-X) installs the patch file so that other users can load it. This command saves and compiles the patch file (patches are always compiled). If the compilation produces compiler warnings, the command asks whether or not you want to finish the patch anyway. If you do, or if no warnings are produced, a new version of the patch directory file is written. The in-progress mark is removed from the entry in the patch registry.

The command allows you to edit the patch comments, which are written to the patch directory file. (*load-patches* and *zk:print-system-modifications* print these comments.) It then asks you whether you want to send mail about the patch. If you say "yes", it opens a mail buffer and inserts initial contents, including the name of the patch file and your patch comment.

Note: By default the *Finish Patch* command queries you about sending mail. You can alter this behavior by changing the value of the variable *zwei:*send-mail-about-patch**. Its valid values are *:ask*, the default value, which queries the user; *t*, which opens a Zmacs mail buffer without querying; and *nil*, which takes no action regarding the sending of patch mail.

Sometimes you start making a patch file and for a variety of reasons do not finish it – for example, you decide to abort the patch, you need to end your work session at this machine, or your machine crashes. In each of these situations it is of the utmost importance that you leave the patch directory file in a clean state; that is, either go back and finish the patch (as soon as possible!) or deallocate the patch number reserved to you. Failure to do so has unfortunate consequences: users at your site will not be able to load patches.

If your machine has crashed, use *Resume Patch* (m-X) to reclaim access to the patch number previously assigned to you. You can continue with the patch (assuming you saved the source files just prior to the crash) or use *Abort Patch* (m-X) to deallocate the patch number. Begin the patch again if you wish. If you simply decide to abandon the patch file, then just use *Abort Patch*. If you must boot your machine before finishing the patch, then save the patch buffer and as soon as possible use *Resume Patch* to read in the relevant patch file; finish the patch or abort it, as you wish.

Patching, *cont'd.*

Abort Patch (m-X)

Abort Patch (m-X) deallocates the minor version number that was assigned by the Start Patch or Add Patch commands. It tells Zmacs that you are no longer interested in making the current patch and offers to kill the patch buffer. The next time you do Add Patch (m-X), Zmacs starts a new patch instead of appending to the one in progress.

Resume Patch (m-X)

Resume Patch (m-X) allows you to return to a patch that you were not able to finish in the same boot session in which you started it; for example, your machine might have crashed or you had to boot your machine suddenly. It reads in the relevant patch file if it was previously saved; otherwise it just reclaims your access to the minor version number allocated to you when you started the patch. Abort or finish the patch.

Under certain circumstances you might find it necessary to recompile and reload a patch file.

Recompile Patch (m-X)

Recompile Patch (m-X) recompiles an existing patch file. This command is useful when, for example, an existing patch needs to be edited or a compiled patch file becomes damaged in some way. Never recompile a patch manually or in any way other than using the Recompile Patch command. This command ensures that source and object files are stored where the patch system can find them.

Use Recompile Patch with caution! Recompiling a patch that has already been loaded by other users can cause divergent world loads.

Reload Patch (m-X)

Reload Patch (m-X) reloads an existing patch file. This command makes it easy to reload a patch file without having to know its pathname.

You might want to have your herald announce private patches

Patching, *cont'd.*

that you make. **note-private-patch** adds a private patch to the database in your world and includes the name of the patch in the herald.

note-private-patch *string**Function*

Adds a private patch to the database in your world. **note-private-patch** takes a *string* argument. For example, the following adds the private patch called patch.lisp:

```
(note-private-patch "s:>smiller>patch.lisp")
```

Subsequent displays of your herald show the inclusion of that patch in your world.

You create private patches using the Start Private Patch (M-X) command and then the standard patch commands for adding to and finishing the patch. Use the Load File command or Load File (M-X) to load a private patch; the load patches command and the **load-patches** function do not load private patches.

15. Customizing the Zmacs Environment

Overview

Introduction

Now that you are familiar with the basic Zmacs concepts and techniques, you can set up a large set of minor modes, Zmacs and Lisp variables, and parameters to change the way the editor works. Zmacs's flexibility allows you to change which keys are connected to which commands, write your own commands, and install them in lieu of the standard system commands. A few users make extremely radical changes to the point where almost every key has a new meaning.

This section describes:

- Zmacs minor and major modes, and how they provide a degree of customization
 - Creating new commands with keyboard macros
 - Saving keyboard macros
 - Setting key bindings
 - Specifying Zmacs variable settings
 - Sample init file forms for automatically reloading your customized environment
-

Built-in Customization Using Zmacs Minor Modes

Definition of Minor Modes

A *minor mode*:

- Is an option.
- Is independent of other minor modes and of the selected major mode.

How It Works

Zmacs has an extended command for each minor mode (*m-X*) that turns the mode on or off. With no argument, the command turns the mode on if it was off and off if it was on. This is known as *toggling*. A positive argument always turns the mode on, and a zero argument or a negative argument always turns it off.

All the minor mode commands are suitable for connecting to single- or double-character commands if you want to enter and exit a minor mode frequently. See the section "Zmacs Key Bindings", page 267.

For information about setting minor modes permanently: See the section "Setting Mode Hooks in Init Files", page 273.

Example

Auto Fill Mode (*m-X*)

Turns on *Auto Fill Mode*, a minor mode that inserts Return characters automatically to break lines as you type. You can turn Auto Fill Mode on regardless of your major mode. If the mode line displays `Fill`, Auto Fill Mode is on. If Auto Fill Mode is already turned on, this command turns it off.

This mode is useful when you are typing large amounts of text. It makes it unnecessary to look at the screen or to worry about line length: you just type in the text without newlines and Zmacs inserts them whenever they are needed.

Auto Fill Mode works by establishing a hook that runs after you press one of the activation characters (SPACE, RETURN, ., ?, !, or |) that activate filling in this mode. When you press one of these characters in Auto Fill Mode, Zmacs does more than simply insert it. First it checks to see whether the line exceeds the maximum allowable line length or *fill column* (see Set Fill Column below). If the line is too long, Zmacs finds the last word on the current line that fits inside the fill column. Zmacs then inserts a newline right after that word. Extra spaces (if any) are deleted from the beginning of the newly formed line.

Built-in Customization Using Zmacs Minor Modes, *cont'd.*

Because of the way Auto Fill Mode works, you will often find yourself typing a word out beyond the fill column. The word will be moved to the next line as soon as you press one of the activation characters.

The fill column is used by Auto Fill Mode (and by the paragraph adjusting commands) to decide where to break lines. It is measured in pixels, not in characters, so that Auto Fill Mode works even if characters of different widths appear in a buffer. (A *pixel* is a tiny rectangular area on the screen that is either all white or all black. Pixels are the smallest addressable region of the display. If you look closely, you can see the separate rectangular pixels that make up everything on the display.)

You can change the fill column with the following command:

`c-X F` Set Fill Column

Changes the fill column to match up with the current position of the cursor. That means that if point is at the end of a line, filled lines will not be longer than the current one from now on. You are given a choice of whether to set the fill column for the buffer, the major Zmacs mode, or globally.

With a positive numeric argument n less than 200, the fill column is set to be n character-widths, and if n is 200 or greater, the fill column is set to be n pixels.

In effect, this command sets the right margin. Auto Fill mode and the `m-Q` use the fill column setting.

Summary of Minor Modes

Atom Word Mode (`m-X`)

Makes word-moving commands, in Lisp mode, move over Lisp objects (other than lists and `nil` instead of words. This command does not display anything in the mode line.

Auto Fill Lisp Comments Mode (`c-m-X`)

Turns on auto filling of comments, but not code. This command displays `Fill-Comments` in the mode line.

Auto Fill Mode (`m-X`)

Turns on auto filling. Auto Fill mode allows you to type text endlessly without worrying about the width of your screen. Return characters are inserted where needed to prevent lines

Built-in Customization Using Zmacs Minor Modes, *cont'd.*

from becoming too long. This command displays `Fill` in the mode line.

Electric Font Lock Mode (m-X)

Puts comments in font B. This command displays `Electric Font-lock` in the mode line.

Electric Shift Lock Mode (m-X)

Facilitates typing in programs that are in uppercase. Whenever you type a character that is part of a Lisp symbol, such as the name of a function, variable, or special form, Zmacs inserts it in uppercase, but when you type a character that is part of a character string or a comment or after a slash, Zmacs inserts it normally. This command displays `Electric Shift-lock` in the mode line.

EMACS Mode (m-X)

Provides commands for EMACS users. It puts bit-prefix commands on `ESCAPE`, `c-^`, and `c-C`, and Universal argument on `c-U`. It also makes `c-I` a synonym for `TAB`, `c-H` a synonym for `BACKSPACE`, and `c-]` a synonym for `ABORT`. This command displays `EMACS` in the mode line.

Overwrite Mode (m-X)

Turns on overwrite mode. In overwrite mode, ordinary printing characters replace existing text, instead of inserting themselves next to it. It is good for editing pictures. This command displays `Overwrite` in the mode line.

Word Abbrev Mode (m-X)

Allows you to define word abbreviations that expand as you type them. This command displays `Abbrev` in the mode line.

Major Modes

User-Defined Major Modes

In Zmacs, you can define your own major modes (see `zwei:defgroup` in the code).

File Types and Major Modes

You can control the default major mode associated with a particular file type. For example, Zmacs sets the major mode to Lisp for files with type `lisp`. The repository for this information is a list called `fs:*file-type-mode-alist*`.

For example, suppose you wanted to associate the file type `tex` with text mode:

```
(push '("tex" . :text) fs:*file-type-mode-alist*)
```

The `car` of an element should be either a canonical type symbol or a string when the type is not one of the known canonical types.

In addition, suppose you have files that would require Scribe mode, if Zmacs had such a thing. You can define a correspondence between two major modes, using a global variable called `zwei:*major-mode-translations*`. It is an alist of major mode names, expressed as keyword symbols.

Example:

```
(push '(:scribe . :text) zwei:*major-mode-translations*)
```

Creating New Commands with Keyboard Macros

Definition

A *keyboard macro* is a command that you define to abbreviate a sequence of other commands. If you discover that you are about to type `c-N c-D` 40 times, you can define a keyboard macro to do `c-N c-D` and call it with a repeat count of 40.

The keyboard macros described here are temporary keyboard macros; that is, macros that you use for the duration of your Zmacs session. For information on writing permanent keyboard macros that you can save and initialize in your init file: See the section "Writing and Saving Keyboard Macros", page 259.

How It Works

You define a keyboard macro by telling Zmacs that you are about to write a macro and then typing the commands that are the definition. That is, as you are defining a keyboard macro, the definition is being executed for the first time. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

Procedure

1. To start defining a keyboard macro, type `c-X (` (Start Kbd Macro). From then on, your commands continue to be executed, but also become part of the definition of the macro. `Macro-level: 1` appears in the mode line.
2. If you want to perform an operation on each line, do one of the following:
 - Start by positioning point on the line above the first one to be processed and then begin the macro definition with a `c-N`
 - Start on the proper line and end with a `c-N`.
 Either way, repeating the macro operates on successive lines.
3. After defining the body of the macro, you can terminate it in several ways.
 - `c-X)` (End Kbd Macro) terminates the definition.
 - An argument of zero to `c-X)` automatically repeats the macro (upon termination of the definition) until it gets an error or reaches the end of the buffer.
 - `c-X)` can be given a repeat count as a numeric argument, in which case it repeats the macro that many times right after

Creating New Commands with Keyboard Macros, *cont'd.*

defining it, but defining the macro counts as the first repetition (since it is executed as you define it). (Subsequent invocations ignore the numeric argument contained in the macro.)

Inserting an argument of 5 before ending the macro (...c-5 c-X) executes the macro immediately four additional times.

Starting a Keyboard Macro

c-X (Start Kbd Macro

Begins defining a keyboard macro. A numeric argument means append to the previous keyboard macro.

Ending a Keyboard Macro

c-X) End Kbd Macro

Terminates the definition of a keyboard macro.

Showing a Keyboard Macro

To see the keyboard macro, use Show Keyboard Macro (m-X), which prints the macro at the top of your screen.

Show Keyboard Macro (m-X)

Displays the specified keyboard macro. The name of the macro is read from the minibuffer; just RETURN means the last one defined, which can also be temporary.

Calling the Last Keyboard Macro

The macro thus defined can be invoked again with c-X E (Call Last Kbd Macro), which can be given a repeat count as a numeric argument to execute the macro many times.

c-X E Call Last Kbd Macro

Repeats the last keyboard macro.

Example

The example below defines a keyboard macro that goes to the beginning of a line, inserts a semicolon, and goes to the next line. It also executes the macro four times, including once as it is being defined.

Creating New Commands with Keyboard Macros, *cont'd.*

```
c-X (
c-A
;
c-N
c-4 c-X )
```

For information about setting key bindings permanently: See the section "Zmacs Key Bindings", page 267.

Writing and Saving Keyboard Macros

Writing and saving keyboard macros entails:

- Defining the macro with **zwei:define-keyboard-macro**.
- Installing the macro on a keystroke with **zwei:make-macro-command**.
- Storing the macro into a comtab with **zwei:command-store**.

zwei:define-keyboard-macro takes as its arguments the name of the macro and the keystrokes specifying what you want it to do.

Optionally, you can install the macro on a keystroke with **zwei:make-macro-command**, giving the name of the macro, which returns a Lisp function.

zwei:command-store takes that Lisp function and stores it into a comtab, similar to what **zwei:set-comtab** does.

zwei:command-store, given the key you want to install the macro on and the comtab in which to put it, stores the command in the slot of the comtab that you specify. The combination of **zwei:make-macro-command** and **zwei:command-store** does the same thing as the Install Macro (M-X) command.

Using variations of the following forms you can save the macros on disk and, if you wish, edit them.

Creating New Commands with Keyboard Macros, *cont'd.*

Example 1

Suppose you want to have a command that exchanges the first two words on a line. Put this form in your init file:

```
(ZWEI:DEFINE-KEYBOARD-MACRO EXCH-FIRST-TWO-WORDS (NIL)
  #\C-A #\M-F #\M-T)
```

The macro cannot be more than 255 keystrokes long. If your macro gets this long you should be writing in Lisp, since keyboard macros are not intended to be a programming language. If necessary, you can get around this restriction by breaking your macro into parts and having them call each other.

Suppose you want to install the EXCH-FIRST-TWO-WORDS macro on the keystroke s-Q. Put this form in your init file:

```
(ZWEI:COMMAND-STORE (ZWEI:MAKE-MACRO-COMMAND 'EXCH-FIRST-TWO-WORDS)
  #\S-Q ZWEI:*ZMACS-COMTAB*)
```

Example 2

The following form defines a keyboard macro called replace-test, which replaces the string dog with the string river:

```
(ZWEI:DEFINE-KEYBOARD-MACRO REPLACE-TEST (NIL)
  #\C-S "dog" #\ESCAPE #\M-RUBOUT "river")
```

To save the keyboard macro replace-test on the keystroke h-S:

```
(ZWEI:COMMAND-STORE (ZWEI:MAKE-MACRO-COMMAND 'REPLACE-TEST)
  #\H-S ZWEI:*ZMACS-COMTAB*)
```

The h-S command takes a numeric argument as a repeat count.

Defining an Interactive Keyboard Macro

Within the keyboard macro definition, you can specify steps at which you want the macro to query. To define an interactive keyboard macro, use the Kbd Macro Query command after beginning the macro definition (with Start Kbd Macro). Invoke Kbd Macro Query at each spot in the macro where you want the macro to query. Then close the definition with End Kbd Macro.

Creating New Commands with Keyboard Macros, *cont'd.*

<code>c-X Q</code>	Kbd Macro Query
Allows user interaction on each iteration of macro, similar to Query Replace (<code>m-X</code>). While defining a keyboard macro, press <code>c-X Q</code> at each step where you want a pause to occur. Upon execution of the macro, it stops and waits at each of those steps for one of the following characters:	
SPACE	Continues execution of the macro.
RUBOUT	Skips rest of keyboard macro (use nested <code>c-X (</code> and <code>c-X)</code> for grouping to control range of skip).
? or HELP	Displays HELP information.
.	Continues but does not iterate anymore.
!	Continues, iterates, but does not ask anymore.
<code>c-R</code>	Enters editing mode; <code>c-m-FUNCTION R</code> resumes the keyboard macro.

Naming a Keyboard Macro

Having defined a keyboard macro, you can name it with Name Last Kbd Macro (`m-X`). A prompt (Name for macro:) appears in the minibuffer.

Name Last Kbd Macro (`m-X`)

Assigns a name to the most recent temporary keyboard macro, making it permanent. The new name for the macro is read from the minibuffer.

Using Keyboard Macros to Sort

You can use a keyboard macro to set up a sorting mechanism and run it on any region of text.

For information about how to sort using keyboard macros, see the description of Sort Via Keyboard Macros (`m-X`): See the section "Overview of Sorting in Zmacs", page 128.

Installing a Macro on a Key

To bind the macro to the key of your choice, use Install Macro (`m-X`). You are asked to identify the macro and specify the key(s) to which you want it bound.

Creating New Commands with Keyboard Macros, *cont'd.*

Install Macro (m-X)

Installs a specified user macro on a specified key. The name of the macro is read from the minibuffer, and the keystroke on which to install it is read in the echo area. If the key is currently holding a command prefix (such as c-X), it asks you for another character, so that you can redefine c-X commands. However, with a numeric argument, it assumes you want to redefine c-X itself, and does not ask for another character.

Installing a Mouse Macro

You can bind the macro to a mouse click instead of a key using Install Mouse Macro (m-X). This command works similarly to Install Macro.

Install Mouse Macro (m-X)

Installs a specified user macro on a specified mouse click. The name of the macro is read from the minibuffer, and the mouse click on which to install it is read in the echo area. When the mouse is clicked to invoke this macro, the macro is invoked from the current location of the mouse cursor.

Deinstalling a Macro

To remove the macro from that key, use Deinstall Macro (m-X). The key is rebound to the standard system usage, if any.

Deinstall Macro (m-X)

Deinstalls a keyboard macro.

Example

This example shows how to install a macro and deinstall the same macro:

```

you type:      m-X Install Macro
minibuffer:    Name of macro to install (CR for last macro defined):
you type:      macro-name or CR
minibuffer:    Key to get it:
you type:      h-T

```

Creating New Commands with Keyboard Macros, *cont'd.*

A menu appears and asks you in which comtab to install the macro:

- Just this editor
- Zmacs
- Zwei

Click on your choice.

minibuffer: Command #<DTP-CLOSURE 34465726> installed on Hyper-T.

you type: m-X Deinstall Macro

minibuffer: Key to deinstall:

you type: h-T

The menu appears and asks you to specify in which of the three comtabs to deinstall the macro. Click on your choice.

minibuffer: Command NIL installed on Hyper-T.

For information about saving keyboard macros permanently: See the section "Zmacs Key Bindings", page 267.

Making Tables

Using Keyboard Macros

The keyboard macro facility implemented with the `c-m-FUNCTION` key provides more features, such as an easy way to make tables.

`c-m-FUNCTION`

Reads a keyboard macro command, consisting of an optional numeric argument made up of any number of digits (0-9) followed by a non-numeric character, usually a letter. Each keyboard macro command must be preceded by the `c-m-FUNCTION` prefix. After typing the prefix, you may type `HELP` for a list of available keyboard macro commands.

Keyboard Macro Commands for `c-m-FUNCTION`

- `0-9` Optional numeric argument.
- `C` Calls a macro by name. Prompts in the minibuffer for the name of the macro.
- `P` Begins a macro definition (same as `c-X (` - See the section "Starting a Keyboard Macro", page 258.)

Creating New Commands with Keyboard Macros, *cont'd.*

- R Ends a macro definition (same as `c-X`) – See the section "Ending a Keyboard Macro", page 258.)
- M Defines a named macro. Prompts for the name of the macro to define and then enters macro definition mode.
- S Stops (aborts) macro definition (also `c-G`).
- D Defines a named macro but does not execute it while reading its characters.
- SPACE Inserts pauses for user interaction in the macro (same as `c-X Q` – See the section "Defining an Interactive Keyboard Macro", page 260.)
- A Steps through characters on successive iterations (for example, letters and numbers). Asks for starting character, amount to increase (or decrease if negative) on each iteration.
- U Allows typein terminated by `c-m-FUNCTION R`. This allows you to stop while in the middle of defining the macro, do other things in the editor, and then go back and finish defining the macro.
- T Allows typein every iteration.

The difference between `c-m-FUNCTION U` and `c-m-FUNCTION T` is that `c-m-FUNCTION U` allows typein while defining a macro. This typein does not get stored in the macro, and therefore does not get executed on subsequent iteration, nor when the macro is called again.

`c-m-FUNCTION T` allows typein on every iteration. As with `c-m-FUNCTION U`, the typein while defining the macro does not get stored in the macro. But on each subsequent iteration, new typein will be requested.

Example 1

The following example shows how to create a macro that constructs a table using `c-m-FUNCTION A`.

Creating New Commands with Keyboard Macros, *cont'd.*

```
you type: c-X (
Minibuffer: Macro-level: 1 *
you type: c-n-FUNCTION R
Minibuffer: Initial character (type a one-character string):
you type: a RETURN
Minibuffer: Amount by which to increase it (type a decimal number):
you type: 1 RETURN
                                (Zmacs inserts the a into the buffer.)
you type: c-2 c-6 c-X )
```

As you close the macro, Zmacs inserts into the buffer:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

by executing the macro 26 times, increasing the letter once each time.

Example 2

The following example shows how to create a macro that constructs a table using `c-n-FUNCTION R`, and this time, `c-n-FUNCTION T`, which allows typein during every iteration of the macro:

Creating New Commands with Keyboard Macros, *cont'd.*

```
you type: c-X (
Minibuffer: Macro-level: 1 *
you type: Item SPACE
you type: c-m-FUNCTION R
Minibuffer: Initial character (type a one-character string):
you type: 1
Minibuffer: Amount by which to increase it (type a decimal number):
you type: 1
you type: TAB
you type: c-m-FUNCTION T
Minibuffer: Macro-level: 2 *
you type: Rosemary
you type: c-m-FUNCTION R
Minibuffer: Macro-level: 1 *
you type: RETURN
you type: c-5 c-X )
you type: Sage
you type: c-m-FUNCTION R
you type: Thyme
you type: c-m-FUNCTION R
you type: Parsley
you type: c-m-FUNCTION R
you type: Pepper
you type: c-m-FUNCTION R
```

The table looks like this:

```
Item 1 Rosemary
Item 2 Sage
Item 3 Thyme
Item 4 Parsley
Item 5 Pepper
```

Key Bindings

Definition

A *key binding* is the set of specific keystrokes that invoke a specific command.

How Key Bindings

Work: the Comtab

A *command table*, or *comtab*, assigns a command to each possible keystroke. While Zmacs is running, there is always a unique *selected comtab*, in which Zmacs finds the command that corresponds to each user keystroke.

When you type a keystroke, Zmacs looks up the keystroke in the currently selected comtab, finds the appropriate command, and runs it. Usually the command's side effects all occur within the buffer: Point might be moved and text might be deleted, inserted, or rearranged. Sometimes a command has more extensive side effects. A command can alter or replace the selected comtab itself, in which case Zmacs looks up the next keystroke in the new command table.

Zmacs's *basic state* consists of the standard editor key bindings, which reside in one special command table, the *standard comtab* (*Zwei comtab*). The standard comtab interacts with the Zmacs comtab and the various mode-dependent comtabs. The typical selected comtab when in Zmacs is "unnamed" for mode-specific key bindings, which indirects to "Zmacs", which indirects to "Zwei". Although the standard comtab can be temporarily replaced, it is always reselected eventually, often after only one "nonstandard" keystroke.

A keystroke that functions as a prefix actually runs a command that replaces the standard comtab for one keystroke. This is the mechanism by which multikeystroke commands are implemented. For example, there are many two-stroke commands whose first keystroke is `c- \mathcal{K}` . This keystroke runs a command that brings in its own comtab before interpreting the next stroke.

Setting the Key

If you want to put a command on the keystroke of your choice, use `Set Key`. This command works for any of the already defined commands.

Set Key (`m- \mathcal{K}`)

Installs a specified command on a specified key. If the key is currently holding a command prefix (such as `c- \mathcal{K}`), it asks you for

Key Bindings, *cont'd.*

another character so that you can redefine `c-X` commands. However, with a numeric argument, it assumes you want to redefine `c-X` itself and does not ask for another character.

It assigns key bindings in the editor that are active in all buffers, and takes two arguments: the name of a command, and a keystroke to invoke it. It reads the name of the command in the minibuffer, completing any command name in any comtab.

Install Command

If you want to put a function on the keystroke of your choice, use `Install Command`. It takes a function, regards it as a command, and puts it on a key.

`Install Command (n-X)`

Installs a specified function as a command in the comtab, on a specified key. It takes two arguments: the name of the function (the current definition, that is, top-level expression), and a keystroke to invoke it. (Zmacs treats as a definition any top-level expression having in functional position a symbol whose name begins "def".) If the key is currently holding a command prefix (such as `c-X`), it asks you for another character so that you can redefine `c-X` commands. However, with a numeric argument, it assumes you want to redefine `c-X` itself and does not ask for another character.

How to Specify Zmacs Variable Settings

Definition

A *variable* is a name that is associated with a value, for example, a number or a string. Zmacs has editor variables that you can set for customization. (Variables can also be set automatically by major modes.)

You can distinguish the names of Zmacs variables from other Lisp variables by their names – the first letters are capitalized and the names contain spaces rather than hyphens.

Finding Out About Zmacs Variables

To examine the value of a single Zmacs variable, use Describe Variable (m-x). To print a complete list of all variables, use List Variables (m-x).

Some commands refer to variables that do not exist in the initial environment. Such commands always use a default value if the variable does not exist. In these cases you must create the variable yourself if you wish to use it to alter the behavior of the command.

Describing Zmacs Variables

Describe Variable (m-x)

Displays the documentation and current value for a single Zmacs variable. It reads the variable name from the minibuffer, using completion.

Listing Zmacs Variables

List Variables (m-x)

Lists *all* Zmacs variables and their values. With a numeric argument, this command also displays the documentation line for the variable.

Listing Variables

by Matching a String

HELP V

Variable Apropos

c-HELP V

c-m-? V

Displays the names of all possible Zmacs variables containing a specific substring. With a numeric argument, this command also displays the documentation lines for the variables.

How to Specify Zmacs Variable Settings, *cont'd.*

Example

One example of such a Zmacs variable is the Fill Column variable, which specifies the width, in pixels, used in filling text.

For example, `c-1 HELP V` prompts in the minibuffer Variable Apropos (substring): and you type `fill col`. It does pattern matching on the variable name and thus matches Fill column, which displays: Fill column: 576. Width in pixels used in filling text.

Setting Variables

Settable Zmacs Variables

You can view all settable Zmacs variables with the List Variables command.

The following are some examples of variables that can be set with Set Variable. In addition, they can be set in init files by using the internal form of their names. For example, Region Marking Mode is `zwei:*region-marking-mode*` internally.

Region Marking Mode

Value: `:reverse-video` for setting the region to reverse video. The default is `:underline`.

Region Right Margin Mode

Value: `t`. Causes whatever marks the region (reverse video or underlining) to extend across unfilled space to the right margin. The default is `nil`.

How to Specify Zmacs Variable Settings, *cont'd.*

One Window Default

Controls which window remains selected after a One Window (c-X 1) command when you were using more than one window. Possible values:

:current

:other

:top

:bottom

This feature operates best when the current layout has no more than two windows. The value **:current** is the only one that is always well defined with more than two windows on the screen.

Check Unbalanced Parentheses When Saving

Controls whether Zmacs checks a file for unbalanced parentheses when you are saving the file. The check is on (t) by default. When it checks a file that you are saving and finds unbalanced parentheses, it queries you about whether to go ahead and save anyway. This applies to all major modes based on Lisp; it is ignored for text modes.

Set Variable

Set Variable (m-X)

Sets any existing Zmacs variable. This command reads the name of a variable (with completion), displays its current value and documentation, and prompts in the minibuffer for a new value. It does some checking to see that the new value has the right type.

Although either uppercase or lowercase works, you are encouraged to capitalize each word of the name for aesthetic reasons, since Zmacs stores the name as you give it.

Customizing Zmacs in Init Files

Introduction

As you gain sophistication with the more advanced features, you will find the settings of parameters that most please you and put these into a command file (*init file*) that the system executes every time you log in.

Creating an Init File

Create a file named *lisp-init.lisp* (or with the correct Lisp file type suffix for your host operating system) in your home directory on your host system and put your Zmacs customizations there.

This section contains examples of forms that you can place inside a **login-forms** in your init file to customize the editor.

login-forms is a special form for wrapping around a set of forms in your init file. It evaluates the forms and arranges for them to be undone when you log out.

Setting Editor Variables

The forms described show how to set Zmacs variables (the kind that Set Variable (M-X) sets).

To set these variables, which are symbol macros, you must use the **zl:setf** macro. For a description of symbol macros: See the section "Symbol Macros" in *Symbolics Common Lisp*. For a description of the **zl:setf** macro: See the macro **zl:setf** in *Symbolics Common Lisp*.

Ordering Buffer Lists

```
(SETF ZWEI:*SORT-ZMACS-BUFFER-LIST* NIL)
```

This displays the list of buffers in the order the buffers were created rather than in the order they were most recently visited.

Putting Buffers Into Current Package

```
(SETF ZWEI:*DEFAULT-PACKAGE* NIL)
```

This puts buffers created with c-X B (Select Buffer) into whatever package is current; the default is to put them in the user package.

Customizing Zmacs in Init Files, *cont'd.*

Setting Default Major Mode

```
(SETF ZWEI:*DEFAULT-MAJOR-MODE* ' :TEXT)
```

This sets the default major mode to Text Mode for buffers with no Mode attribute and no major mode deducible from the file type; the default is Fundamental Mode.

Setting Find File Not to Create New Files

```
(SETF ZWEI:*FIND-FILE-NOT-FOUND-IS-AN-ERROR* T)
```

This beeps and prints an error message when you give `c-X c-F` (Find File) the name of a nonexistent file. The default prints (New File) and creates an empty buffer, which when saved by `c-X c-S` (Save File) creates the file that was nonexistent.

Setting Goal Column for Real Line Commands

```
(SETF ZWEI:*PERMANENT-REAL-LINE-GOAL-XPOS* 0)
```

This moves subsequent `c-N` and `c-P` (Down Real Line and Up Real Line) commands to the left margin, like doing `c-0 c-X c-N` (Set Goal Column to zero).

Fixing White Space for Kill/Yank Commands

```
(SETF ZWEI:*KILL-INTERVAL-SMARTS* T)
```

This tells the killing and yanking commands optimize white space surrounding the killed or yanked text.

Setting Mode Hooks

Each major mode has a *mode hook*, a variable which, if bound, is a function that is called with no arguments when that major mode is turned on.

Electric Shift Lock in Lisp Mode

```
(SETF ZWEI:LISP-MODE-HOOK 'ZWEI:ELECTRIC-SHIFT-LOCK-IF-APPROPRIATE)
```

This tells Lisp major mode to turn on Electric Shift Lock minor

Customizing Zmacs in Init Files, *cont'd.*

mode unless the buffer has a Lowercase attribute. The effect is that by default Lisp code is written in upper case.

Auto Fill in Text Mode

```
(SETF ZWEI:TEXT-MODE-HOOK 'ZWEI:AUTO-FILL-IF-APPROPRIATE)
```

This tells Text major mode to turn on Auto Fill minor mode unless the buffer has a Nofill attribute. The effect is that by default lines of text are automatically broken by carriage returns when they get too wide.

Key Bindings

To bind keys, you first define the comtab in which to put the binding. For example, ***standard-comtab*** and ***standard-control-x-comtab*** define features of all Zwei-based editors; ***zmacs-comtab*** and ***zmacs-control-x-comtab*** define features that are Zmacs-specific.

White Space in Lisp Code

```
ZWEI:(SET-COMTAB *STANDARD-CONTROL-X-COMTAB*
          '#\SP COM-CANONICALIZE-WHITESPACE))
```

This defines c-X SPACE as a command that makes the horizontal and vertical white space around point (or around mark if given a numeric argument or immediately after a yank command) conform to standard style for Lisp code.

c-m-L on the SQUARE Key

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
          '#\SQUARE COM-SELECT-PREVIOUS-BUFFER))
```

This defines the SQUARE key to do the same thing as c-m-L. This key binding is placed in ***zmacs-comtab*** rather than ***standard-comtab*** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on c-X c-B

```
ZWEI:(SET-COMTAB *ZMACS-CONTROL-X-COMTAB*
          '#\c-B COM-EDIT-BUFFERS))
```

This makes c-X c-B invoke Edit Buffers rather than List Buffers. This key binding is placed in ***zmacs-control-x-comtab*** rather

Customizing Zmacs in Init Files, *cont'd.*

than ***standard-control-x-comtab*** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on m-X

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
      ()
      (MAKE-COMMAND-ALIST '(COM-EDIT-BUFFERS)))
```

This makes Edit Buffers available on **m-X** in Zmacs (by default it is only available on **c-m-X**).

m-. on m-(L)

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
      ' (#\m-MOUSE-L COM-EDIT-DEFINITION))
```

This makes clicking the left mouse button while holding down the META key do what **m-.** does. Invoking this command from the mouse is convenient when you specify the name of the definition to be edited by pointing at it rather than typing it.

Appendix A. Zmacs Help Command Summary

This section lists the names of the available help commands grouped according to the context in which they are available. The purpose of this section is to summarize the capabilities and to help you determine both the overall contexts for which you can find help and a particular function that might be what you are looking for.

Zmacs Commands for Finding Out About the State of Buffers

Edit Buffers (m-X)
Edit Changed Definitions (m-X)
Edit Changed Definitions Of Buffer (m-X)
List Buffers (c-X c-B)
List Changed Definitions (m-X)
List Changed Definitions Of Buffer (m-X)
List Definitions (m-X)
List Matching Lines (m-X)
Print Modifications (m-X)
Select System as Tag Table (m-X)
Tags Search (m-X)

Zmacs Commands for Finding Out About the State of Zmacs

Apropos (m-X), HELP A
Describe Variable (m-X)
Edit Zmacs Command (m-X)
List Commands (m-X)
List Registers (m-X)
List Some Word Abbrevs (m-X)
List Tag Tables (m-X)
List Variables (m-X)
List Word Abbrevs (m-X)

Zmacs Commands for Finding Out About Lisp

Describe Variable At Point (c-sh-V)
Edit Callers (m-X)
Edit CP Command m-X
Edit Definition (m-.)
Edit File Warnings (m-X)
Function Apropos (m-X)
List Callers (m-X)
List Matching Symbols (m-X)
Long Documentation (c-sh-D)
Multiple Edit Callers (m-X)
Multiple List Callers (m-X)
Quick Arglist (c-sh-A)
Show Documentation (m-sh-D)
Show Documentation Function (m-sh-A)
Show Documentation Variable (m-sh-V)
Where Is Symbol (m-X)

Zmacs Commands for Finding Out About Flavors

Describe Flavor (m-X)

Show Documentation Flavor (m-sh-F)

Edit Combined Methods (m-X)

Edit Methods (m-X)

List Combined Methods (m-X)

List Methods (m-X)

Zmacs Commands for Interacting with Lisp

Break (SUSPEND)
Compile And Exit (m-Z)
Compile Buffer (m-X)
Compile Changed Definitions (m-X)
Compile Changed Definitions Of Buffer (m-X), m-sh-C
Compile File (m-X)
Compile Region (m-X), c-sh-C
Compiler Warnings (m-X)
Edit Compiler Warnings (m-X)
Evaluate And Exit (c-m-Z)
Evaluate And Replace Into Buffer (m-X)
Evaluate Buffer (m-X)
Evaluate Changed Definitions (m-X)
Evaluate Changed Definitions Of Buffer (m-X), m-sh-E
Evaluate Into Buffer (m-X)
Evaluate Minibuffer (ESCAPE)
Evaluate Region (m-X), c-sh-E
Evaluate Region Hack (m-X)
Evaluate Region Verbose (c-m-sh-E)
Load Compiler Warnings (m-X)
Macro Expand Expression (m-X), c-sh-M
Trace (m-X)
Quit (c-Z)

PART II.

Font Editor

16. Font Basic Concepts

On the Symbolics Lisp Machine, characters can be typed out in any of a number of different typefaces. Some text is printed in characters that are small or large, boldface or italic, or in different styles altogether. Each such typeface is called a *font*. A font is conceptually an array, indexed by character code, of pictures showing how each character should be drawn on the screen. The Font Editor (FED) is a program that allows you to create, modify, and extend fonts: See the section "Font Editor", page 283.

A font is represented inside the Lisp Machine as a Lisp object. Each font has a name. The name of a font is a symbol, usually in the `fonts` package, and the symbol is bound to the font. A typical font name is `tr8`. In the initial Lisp environment, the symbol `fonts:tr8` is bound to a font object whose printed representation is something like:

```
#<FONT TR8 234712342>
```

The initial Lisp environment includes many fonts. Usually there are more fonts stored in BFD files in file computers. New fonts can be created, saved in BFD files, and loaded into the Lisp environment; they can also simply be created inside the environment.

If you are loading a font contained in a font file in one of the font directories, the system loads that font the first time you reference it. However, if you are loading a font contained in a file somewhere else in the file system, load that font using the function `fed:read-font-from-bfd-file pathname`, where *pathname* is the pathname of the font file. See the section "Load-bfd Transformation of Defsystem".

*From book
using Fonts
P
o*

The `zl-user:tv` package contains the window system, which includes fonts for screen display (as opposed to fonts for hardcopying).

16.1 Attributes of TV Fonts

Fonts, and characters in fonts, have several interesting attributes.

Character Height Font Attribute

One attribute of each font is its *character height*. This is a nonnegative integer used to figure out how tall to make the lines in a window. Each window has a certain *line height*. The line height is computed by examining each font in the font map, and finding the one with the largest character height. This largest character height is added to the vertical spacing (in pixels) between the text lines

(*vsp*) specified for the window, and the sum is the line height of the window. The line height, therefore, is recomputed every time the font map is changed or the *vsp* is set. This ensures that any line has enough room to display the largest character of the largest font and still leave the specified vertical spacing between lines. One effect of this is that if you have a window that has two fonts, one large and one small, and you do output in only the small font, the lines are still spaced far enough apart to accommodate characters from the large font. This is because the window system cannot predict when you might, in the middle of a line, suddenly switch to the large font.

Baseline Font Attribute

Another attribute of a font is its *baseline*. The baseline is a nonnegative integer that is the number of raster lines between the top of each character and the base of the character. (The base is usually the lowest point in the character, except for letters that descend below the baseline, such as lowercase p and g.) This number is stored so that when you are using several different fonts side-by-side, they are aligned at their bases rather than at their tops or bottoms. So when you output a character at a certain cursor position, the window system first examines the baseline of the current font, then draws the character in a position adjusted vertically to make the bases of the characters all line up.

Character Width Font Attribute

The *character width* can be an attribute either of the font as a whole, or of each character separately. If there is a character width for the whole font, it is as if each character had that character width separately. The character width is the amount by which the cursor position should be moved to the right when a character is output on the window. This can be different for different characters if the font is a variable-width font, in which a W might be much wider than an i. Note that the character width does not necessarily have anything to do with the actual width of the bits of the character (although it usually does); it is merely defined to be the amount by which the cursor should be moved.

Left Kern Font Attribute

The *left kern* is an attribute of each character separately. Usually it is zero, but it can also be a positive or negative integer. When the window system draws a character at a given cursor position, and the left kern is nonzero, the character is drawn to the left of the cursor position by the amount of the left kern, instead of being drawn exactly at the cursor position. In other words, the cursor position is adjusted to the left by the amount of the left kern of a character when that character is drawn, but only temporarily; the left kern only affects where the single character is drawn and does not have any cumulative effect on the cursor position.

Fixed-width Font Attribute

A font that does not have separate character widths for each character and does not have any nonzero left kerns is called a *fixed-width* font. The characters are all the same width and so they line up in columns, as in typewritten text. Other fonts are called *variable-width* because different characters have different widths and things do not line up in columns. Fixed-width fonts are typically used for programs, where columnar indentation is used, while variable-width fonts are typically used for English text, because they tend to be easier to read and to take less space on the screen.

Blinker Width and Blinker Height Font Attributes

The *blinker width* and *blinker height* are two nonnegative integers that tell the window system an attractive width and height to make a rectangular blinker for characters in this font. These attributes are completely independent of all other attributes and are only used for making blinkers. Using a fixed width blinker for a variable-width font causes problems; the editor actually readjusts its blinker width as a function of what character it is on top of, making a wide blinker for wide characters and a narrow blinker for narrow characters. The easiest thing to do is to use the blinker width as the width of the blinker. This works well with a fixed-width font.

Chars-exist-table Font Attribute

The *chars-exist-table* is `nil` if all characters exist in a font, or an `sys:art-boolean` array. This table is not used by the character-drawing software; it is for informational purposes. Characters that do not exist have pictures with no bits "on" in them, just like the Space character. Most fonts implement most of the printing characters in the character set, but some are missing some characters.

16.2 Standard TV Fonts

You can use `Show Font HELP` in the Lisp Listener or the `List Fonts (m-k)` command in Zmacs to get a list of all the fonts that are currently loaded into the Lisp environment. The `zl-user:fonts` package contains the names of all fonts. Here is a list of some of the useful fonts:

<code>fonts:cptfont</code>	This is the default font, used for almost everything.
<code>fonts:jess14</code>	This is the default font in menus. It is a variable-width rounded font, slightly larger and more attractive than <code>medfnt</code> .

fonts:cptfonti	This is a fixed-width italic font of the same width and shape as fonts:cptfont , the default screen font. It is most useful for italicizing running text along with fonts:cptfont .
fonts:cptfontcb	This is a fixed-width bold font of the same width and shape as fonts:cptfont , the default screen font.
fonts:medfnt	This is a fixed-width font with characters somewhat larger than those of zl-user:cptfont .
fonts:medfntb	This is a bold version of zl-user:medfnt . When you use Split Screen, for example, the [Do It] and [Abort] items are in this font.
fonts:hl12i	This is a variable-width italic font. It is useful for italic items in menus; Zmail uses it for this in several menus.
fonts:tr10i	This is a very small italic font. It is the one used by the Inspector to say " <i>More above</i> " and " <i>More below</i> ".
fonts:hl10	This is a very small font used for nonselected items in Choose Variable Values windows.
fonts:hl10b	This is a bold version of zl-user:hl10 , used for selected items in Choose Variable Values windows.

17. Entering and Leaving FED

You can enter FED:

- By using [Font Edit] in the System menu.
- By typing the Edit Font command at any Lisp Listener.
- By typing the `zl:fed` Lisp form at any Lisp Listener.

The first time you invoke FED in a session, it takes about 15 seconds to start up; after that, entering FED is very quick. When the startup is complete, you see a *FED Frame*, the window configuration used by FED. You are not editing any particular font: you can experiment with character drawing in this state, but it is best to select a font first.

If you know which font you wish to edit before entering FED, you can save time and steps by typing the *font-name* as an argument to Edit Font or `zl:fed`:

Edit Font *font-name*

or

(fed *font-name*)

font-name can be a string, a BFD object, or any atomic symbol (on any package) whose print name is the name of the font you wish to edit.

You can exit FED either by selecting some other activity (via the System menu, mouse, the SELECT key, or FUNCTION S), or by using [EXIT] in FED's menu.

Whenever you reinvoke FED in the same session, you return to the editing that you were doing when you left FED. Thus, only one FED exists per session, and you do not lose your work by leaving it.

Should FED become unusable because of an error, you can type the following form at a Lisp Listener:

(fed :reinitialize)

This creates a completely new FED (although not destroying the old one).

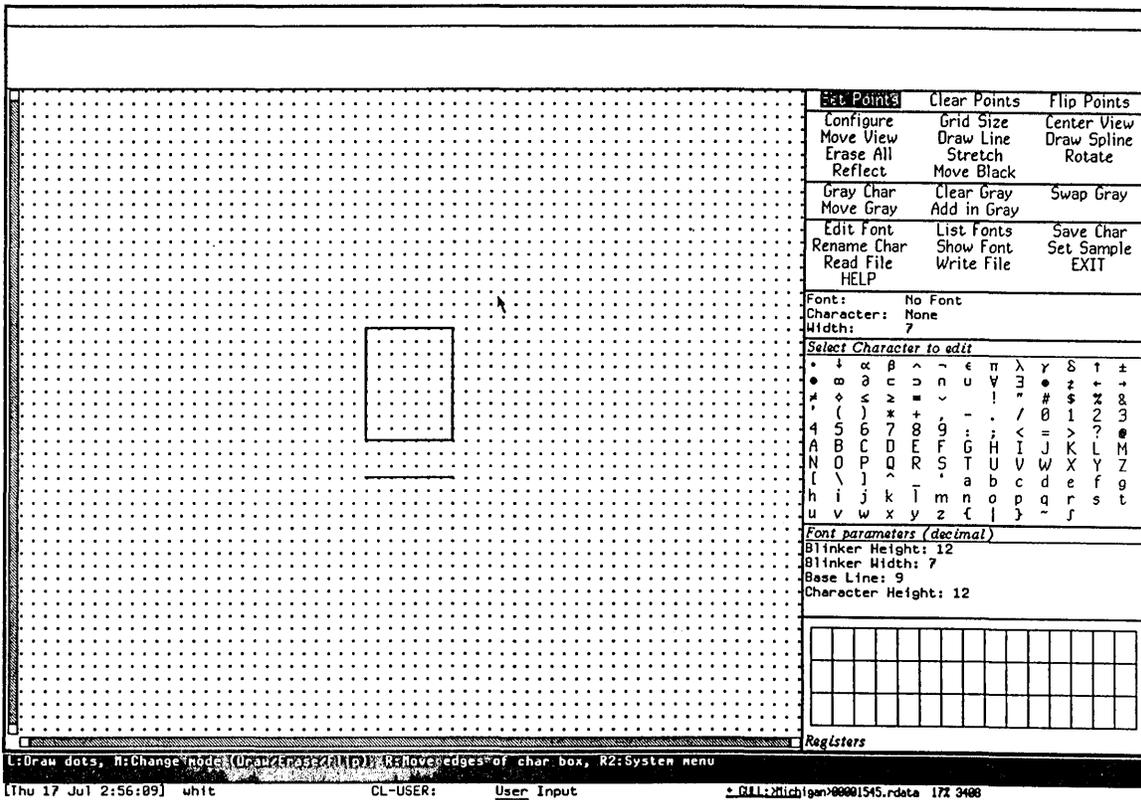


Figure 1. Initial FED Display

18. Font Editor Basic Concepts

18.1 FED, the Subsystem

FED accepts both menu commands and character (keyboard) commands.

When you enter FED, you see a complex *frame* of many *panes*. The following are descriptions of the panes in the FED frame:

Drawing Pane The largest pane is the *drawing pane*, which contains a grid of dots forming an array of squares, and a box drawn in the middle. When you edit a character, FED draws the character in this pane, magnified 12 to 1. (You can choose other magnifications with the [Configure] and [Grid Size] FED menu commands.) Each box, delineated by four dots, represents one *pixel* (bit-raster dot) of the character being edited.

The basic technique of editing characters is to draw lines, points, and curves on this pane, using the mouse as a graphic input device, and thus modify the bit-raster definition of the character being edited. Mouse clicks on the drawing pane draw and clear points. For information on mouse use on the drawing pane: See the section "Drawing in FED", page 299.

Character Box The box drawn in the center of the drawing pane is called the *character box*. It shows the font baseline and character height, as well as the width and kerning of the character being edited. The box itself shows the right and left margins of the character, and the top and baseline of the font. The line under the character box shows the *character height* of the font, which is the height that the window system uses to compute line spacing for windows with the current font in their font map. It represents, in essence, the maximum height of any character in the font, although it is a font parameter, not one computed by inspecting all characters in a font.

You can alter the positioning of the character box, as well the character width it represents. See the section "Viewing and Altering a Character in the FED Character Box", page 301.

Sample Pane The topmost pane of the FED frame is called the *sample pane*. It shows what the character being edited looks like in normal size. That display appears in the leftmost part of the sample pane. About an inch to the right of that, the sample pane

shows a life-size *sample string* in the font being edited. (You can set this sample string with the [Set Sample] FED menu command.) The sample string allows you to see what a given word or phrase, drawn in the font being edited, looks like. This allows you to see your changes to a given character in context. Note that the sample pane changes size as you select fonts of differing character height.

Prompt Pane Between the sample pane and the drawing pane is the *prompt pane*. This is used whenever keyboard typein is required. Occasionally, messages and instructions to you (such as how to use the mouse for curve and line drawing) appear there too.

Menus To the right of the drawing pane is a set of menus and miscellaneous panes.

Draw Mode Menu

The topmost menu is called the *draw mode menu*; it tells the default interpretation of mouse clicks on the drawing pane. One element of the draw mode menu is always highlighted, and specifies the current interpretation of the mouse on the drawing pane. Selecting (by mouse click) any item on the draw mode pane makes the selected mode be the new default, and highlights that mode. Other ways of changing the draw mode also update the highlighting in this pane. See the section "Drawing in FED", page 299.

Under the draw mode menu appear three *command* menus that display a repertoire of commands that you can issue at any time by clicking on their items with the mouse. Many of the items interpret the different mouse buttons differently. See the section "FED Command List", page 327. The mouse documentation line at the bottom of the screen displays the interpretation of the mouse buttons when the mouse is positioned over a potential choice.

The three command menus are grouped by related function:

Drawing Pane Menu

The topmost command menu (drawing pane menu) presents a group of commands allowing you to control how you are looking at what you see, and commands to perform automatic transformation and drawing on the character being edited.

Gray Plane Menu

The second command menu contains commands apropos the *gray plane*, which is, in effect, a second pane behind the drawing pane, whose display is shown in gray instead of black. You can use the gray plane to see two characters at once, to see one character as a model while editing another, and so on. The gray plane can be moved around and manipulated in several ways. See the section "The FED Gray Plane", page 303.

Outside FED Command Menu

The third command menu contains commands dealing with the world outside FED: reading and writing files, getting help, leaving FED, and selecting and saving characters and fonts.

Status Pane

Under the command menus is the *status pane*, which tells you what font and what character is being edited. The character is displayed in the default Lisp Machine font: this is to be considered an *identification* of the character you are editing. For example, if you display the font `zl-user:greek9` with [Show Font], you see that the omega character in `zl-user:greek9` occupies the position that corresponds to W in the default Lisp Machine font. So the status pane identifies that character as W, but the "real" character (omega) is displayed in the sample pane. The status pane also shows you the width of the character being edited. The width is changed by manipulating the vertical edges of the character box; this action updates the status pane's display.

Character Select Menu

Under the status pane is the *Character Select* menu, which is used to select a character to edit. Simply clicking on an item in this menu (once a font has been selected) draws that character in magnification in the drawing pane, so you can begin editing it. You can also use the Character Select menu to answer any prompt for a character, such as those issued by the Rename Character and Gray Character commands. When a prompt is issued that can be answered by clicking on this menu, it says so in its text.

Font Parameters Menu

Under the Character Select menu is the *Font Parameters* menu. It displays the font-wide parameters, such as blinker height and width, and baseline and character height. This is a Choose

Variable Values menu; by clicking on any of the numbers in it, the menu "opens up" and allows you to type in a new value. When you change a font parameter in this way, the change takes effect immediately. The FED frame can even change shape to accommodate the new parameters. All values in this menu are displayed and accepted in decimal, regardless of the setting of `zl:base` and `zl:ibase`.

Register Pane The final pane of the Fed frame is the *register pane*, which is labelled *Registers*. It is divided into as many little boxes (*registers*) as fit; the size of the boxes is computed from the parameters of the current font. Registers can be used to store characters and pieces of characters being edited, and retrieve them, without storing them into any font. See the section "Saving Characters and Pieces of Characters in FED Registers", page 307.

FED has an alternative *configuration*, or pane layout, that gives a wide aspect ratio (screen-wide) to the drawing pane, as opposed to the normal tall aspect ratio. The [Configure] menu item in the top command menu can be used to switch configurations. When selected, it pops up a menu of the two possible configurations.

Many FED commands produce *typeout*, text and/or drawings that are "written over" the whole FED frame display. [Show Font] and [List Fonts] are typical of such commands. When a command produces typeout, the typeout remains until the next command is typed. Pressing SPACE is a command that does nothing; use it to erase typeout and do nothing more.

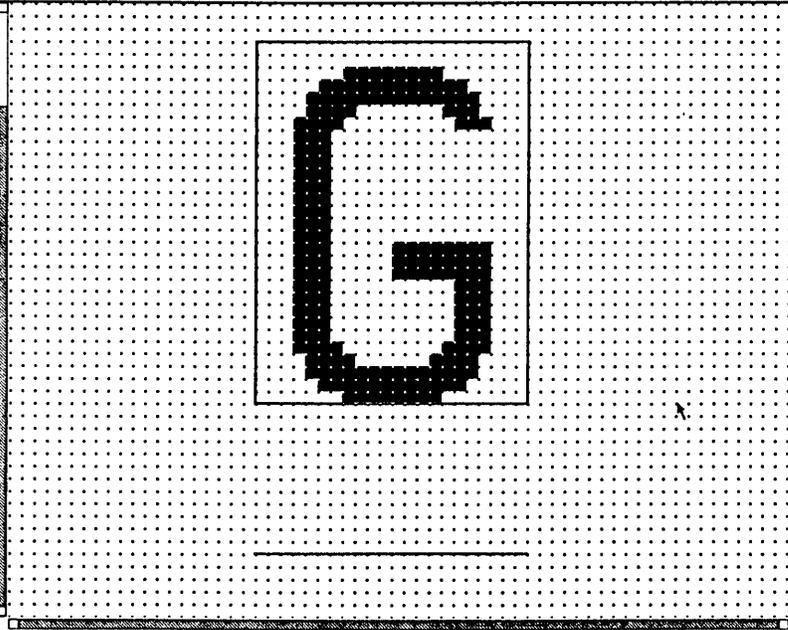
18.2 Selecting a Font

FED edits one font at a time, and one character in that font at a time. You can make new fonts, and add new characters to fonts. Using FED consists of selecting a font, then selecting, successively, several characters in that font, editing each one in turn, and "storing" it back into the font. When this editing is finished, the font in the Lisp environment reflects all of these changes. At that time, you usually want to write the font out to a BFD file, to save your work. See the section "Reading and Writing FED Files", page 325.

FED provides several ways to select a font.

- You can name the font to be edited in the command or Lisp form that invoked FED. See the section "Entering and Leaving FED", page 289.
- You can select the [Edit Font] menu item, which prompts for the font name in

G



Set Points	Clear Points	Flip Points
Configure	Grid Size	Center View
Move View	Draw Line	Draw Spline
Erase All	Stretch	Rotate
Reflect	Move Black	
Gray Char	Clear Gray	Swap Gray
Move Gray	Add in Gray	
Edit Font	List Fonts	Save Char
Rename Char	Show Font	Set Sample
Read File	Write File	EXIT
HELP		

Font: LWFIX18
 Character: G (107)
 Width: 22

Select Character to edit

•	!	α	β	^	~	ε	π	λ	γ	δ	τ	±
•	∞	∂	c	∩	n	u	v	∩	∩	∩	∩	∩
#	∅	≤	≥	≡	≠	!	/	#	\$	%	&	'
4	()	*	+	-	.	/	0	1	2	3	
A	5	6	7	8	9	:	;	<	=	>	?	•
B	C	D	E	F	G	H	I	J	K	L	M	Z
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
l	\]	^	~	'	a	b	c	d	e	f	g
h	i	j	k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z	{		}				

Font parameters (decimal)
 Blinker Height: 42
 Blinker Width: 22
 Base Line: 29
 Character Height: 41

Registers

--	--	--	--	--	--	--	--	--	--	--	--	--

L:Draw dots, R:Change node (Draw/Erase/Flip), R:Move edges of char box, R2:System menu

[Thu 17 Jul 9:00:51] whit CL-USER: User Input • G11:2:Richlan>00001546.rdata 212 48048

Figure 2. Tall Configuration

the prompt pane. Use [Edit Font (M)] to copy an existing font as the first step of making a new font.

You can list all loaded fonts with the [List Fonts] menu item. The display produced by [List Fonts] is mouse sensitive: moving the mouse over the name of any font highlights it, and clicking on it begins editing of that font. Using [List Fonts (R)] lists all fonts on the file computer as well as loaded ones. This usually takes a long time to produce. The keyboard command F can also be used to prompt for the name of a font to edit.

18.2.1 Creating a New Font

If you attempt to edit a font that is not known to the system, FED asks you whether you wish to create that font. This is the way you create new fonts. When you create a new font, the first thing you usually want to do is alter the font parameters (in the font parameters menu) and define the Space character, from which many facilities in the system (including some in FED) determine the "usual width" of characters in this font. As a matter of fact, you might want to reconfigure the FED frame after setting the width of Space, to correctly recalculate the width of registers in the register pane.

18.2.2 Displaying Characters in the Font

When you start editing a font, you are not editing any character. The drawing pane displays a typical character box, and no points. The specifications for the character box reflect the Space character in the font. You must then select a character to edit. FED then displays all of the characters in the font, using the display normally obtainable by [Show Font]. You can erase this display by pressing SPACE or by selecting a character to edit. See the section "Selecting a Character in FED", page 297.

18.3 Selecting a Character

Once a font has been selected, FED edits one character at a time. You modify the definition of the character by drawing and clearing points on the drawing plane. When you are done editing a character, you store it back in the font by using the [Save Char] menu item. Your changes to the character are not saved until you do this. Furthermore, none of your changes to a font being edited become permanent until you write the font out to a file.

18.3.1 From the Character Select Menu

The usual way to select the character being edited is by using the mouse to select a character in the Character Select menu. When you select a character, it is drawn in magnification in the drawing pane, and the status pane is updated to tell you what character you are editing.

18.3.2 By Creating a New Character

If you attempt to edit a character that is not in the font being edited, FED creates a new character. This is the way new characters are created. The new character is not actually saved in the font until the [Save Char] command is issued.

18.3.3 From the [Show Font] Display

You can also select a character by displaying all of the characters of the font being edited, via the [Show Font] menu item. The display produced by this command is mouse sensitive: when you move the mouse over the image of a character, it is highlighted, and if you click on it, editing of that character begins. This display is produced automatically when you select a font to be edited.

18.3.4 With the C Command

The keyboard C command can also be used to select a character. Pressing C prompts for a character, which can be supplied from the keyboard or the Character Select menu.

18.3.5 By Renaming Characters

Another way to edit a character is to *rename* the character being edited to some other character. This is one way to move characters around in a font, and make characters into other characters. Selecting the [Rename Char] menu item prompts for a character to call the character being edited. You can answer this prompt either by typing a character from the keyboard, or from the Character Select menu. This changes FED's idea of what character you are editing, and the status pane and sample string (if any) are updated to reflect this fact. Renaming a character does *not* store it back in the font; you must do that by yourself, as usual, with the [Save Char] command when you are done editing it.

19. Drawing

The most common technique for creating and editing characters is to draw and clear points on the drawing pane using the mouse.

A *nonmouse cursor* can be moved around with the keyboard. Sometimes, as when square-counting is necessary, this is useful. See the section "The FED Nonmouse Cursor", page 300.

19.1 Drawing Characters with the Mouse

Drawing on the drawing pane is in one of three modes at any time, [Set Points], [Clear Points], or [Flip Points]. The highlighted item in the draw mode menu tells which is in effect. When you click left on a box in the drawing pane, that box is made black (set), or white (clear), or complemented (flip), according to the current draw mode. If you hold the left button down (that is, you do not release it after clicking left on a box) and move it around, you set (or clear or complement) all squares over which you pass. In this way, you can draw curves or pictures, fill in areas, clear old mistakes, and so forth. This is the most common operation in FED, and is called *drawing with the mouse*.

You can change the drawing mode either by selecting another draw mode by clicking on an item in the draw mode menu, or by clicking middle on the drawing pane. Clicking middle rotates through the possible draw modes.

When you draw with the mouse, the sample pane is not updated until you release the left button. (You might want to do this every now and then while drawing with the mouse, just to observe what you have in life-size, and then press the left button again, to continue drawing.)

Often, you might want to "temporarily" change the draw mode, either because the draw mode menu is too distant, or the mouse is not in top shape, or because you really want to change the draw mode for just one or two squares. You can do this while drawing by manipulating the CONTROL and META keys on the keyboard. If you hold down CONTROL alone while drawing, the temporary draw mode becomes [Clear Points] for as long as it is held down. Similarly, META alone sets up [Set Points] mode for as long as it is held down. CONTROL and META together temporarily put the mouse in *pass-over* mode, in which it makes no change to any squares it passes over.

Flip mode is useful for final touch-ups, a click at a time, rather than drawing with the mouse button down. Since it changes any square you click on, it is most useful when you fix up single squares in the final stages of editing a character.

19.2 The Nonmouse Cursor

The nonmouse cursor is an "alternative mouse" that can be used to draw in the drawing pane. It can be useful when the mouse is not in top shape, or when you are doing some design that involves counting squares carefully.

This cursor is normally not visible. It starts out in the upper left-hand corner of the drawing pane. You move it via the `/`, `\`, `[`, and `]` keys, which tell the direction in which to move it. See the section "FED Menu and Keyboard Commands", page 327. When you start moving it, it appears as a smaller, blinking box inside the grid box over which it sits. When you draw with the real mouse, it goes away.

The keyboard command `."` complements the box over which the nonmouse cursor sits.

You can also move the nonmouse cursor in numerically specified movements using specialized commands. See the section "FED Command List", page 327.

20. Viewing and Altering a Character in the Character Box

The character box is the mechanism by which you can view and alter the boundaries of a character being edited. The following is a description of its edges, and instructions for changing them.

20.1 What the Lines Mean

FED displays a *character box* in the drawing pane, to indicate the "boundaries" of the character being edited. These boundaries are not absolute limits outside which the character cannot extend; rather, they are the positions that are to be considered the start and end of this character when it is drawn in use.

Characters in italic fonts and foreign scripts often extend into the "territory" of the previous or next character. Such "incursion" is accomplished by a character's containing points outside its limits.

Left and right edges

The left edge of the character box represents the cursor position at the time the character is drawn in real use. Any points to the left of this are in the "territory" of the previous character. The right edge represents the start of the next character. The distance between the left edge and the right edge is called the *character width*, and specifies the distance by which the window system increments its horizontal cursor position after drawing this character. Points to the right of the right edge of the character box are an incursion into the territory of the next character to the right.

Bottom edge

The bottom edge of the character box (*not* the line under it) represents the *baseline* of the font. The baselines of all characters drawn on a line, in any font, form a continuous line, the normal "bottom" of most characters. Points below the baseline are "descenders".

Top edge

The top edge of the character box represents the top of the character. You cannot put points above the top, but FED lets you draw such points, for you might move them and/or the character box before you save the character. FED warns and asks you what to do if you attempt to save a character that has points above its top edge; this is an error. The distance

between the top edge and the baseline is fixed for any given font (although you can use FED to change the value of that number). If you are making a new font, you should carefully consider this parameter (the font's *baseline*) before generating any characters.

Character height The line below the bottom of the box represents the *character height* of the font, which is the distance between the top edge and this line. This distance, too, is a fixed parameter for any font, although you can use FED to alter it for the whole font. You cannot put points below this line; if you do, they appear in the territory of the *next* line when drawn, and are cleared or overwritten inconsistently. The maximum of the character heights of all fonts in the font map of a window is used to compute the line spacing of a window.

20.2 Altering the Character Box

You can move the edges of the character box on the drawing pane by clicking on them (within one-half box on either side) with the right mouse button. Hold the button down and move the line to where you want it to be, and then release the button.

Moving the character box redefines the orientation of the character, as drawn, with respect to the other characters in the same font.

If you attempt to move the bottom edge, top edge, or character height line, you move them all, and thus move the whole character box vertically. You cannot move them individually because the distances between them are fixed parameters for the font. If you alter these parameters by selecting them in the Font Parameters menu, the character box is altered and redrawn appropriately.

Sometimes, you want to move the whole character box without changing its shape. The easiest way to do this is to move the data being displayed with the [Move Black] menu item. See the section "Transformations on Characters in FED", page 309.

21. The Gray Plane

The gray plane is a "shadow" "behind" the drawing pane that allows you to look at another character in addition to the one you are editing. The character (or piece of a character) in the gray plane shows up in light gray in the drawing pane. Where bits are on in both the gray plane and the character being edited (the *black plane*), a dark gray square is shown.

Frequently, the gray plane is used to hold a character that resembles, or has pieces of, the character being edited, to serve as a guide for drawing the new character. At other times, the gray plane is used to hold a piece of a character, to be merged later into the black plane.

The second of the three command menus is a special menu for commands dealing with the gray plane. It is also possible to fetch previously created patterns into the gray plane from the register pane. See the section "Saving Characters and Pieces of Characters in FED Registers", page 307.

21.1 Getting Things Into Gray

The most common ways of putting drawings into the gray plane are to move the black plane into it and to fetch characters into it. The [Swap Gray] and [Gray Char] menu items do this.

21.1.1 With [Swap Gray]

[Swap Gray] exchanges the black and gray planes; what had been black becomes gray, and what had been gray becomes black. After you use [Swap Gray], you are editing in the black plane what had been in the gray plane, and what you had been editing in the black plane (where all editing is done) is now visible in the gray plane. You can clear the black plane with [Erase All]; [Clear Gray] (in the gray plane menu) clears the gray plane.

You can swap the gray and black plane to bring the gray plane up for editing, to move something you have edited into the gray plane, or to do both at once.

21.1.2 With [Gray Char]

You can bring characters directly into the gray plane. Using [Gray Char] prompts you for a character in the current font to be brought into the gray plane. You can then type the character, or select it in the Character Select menu. The keyboard command G does this, too. The character is placed at the character box.

It does not really matter where the character is placed, though, because before merging it or using it, you can move it to any place in the gray plane by using [Move Gray]. See the section "Merging Characters with the FED Gray Plane", page 304.

You can bring characters from other fonts into the gray plane by using [Gray Char (R)]. A Choose Variable Values menu is presented, offering choices not only of character and font, but of scaling as well. Click on values you wish to change; keep in mind that the [Character] item expects a single character when you use it. Scaling allows you to grow or shrink the character being fetched before bringing it into the gray plane. The numerator and denominator of the scale fraction are displayed and interpreted as decimal numbers. When you are done choosing values for [Gray Char], use [Do It] to bring in the character.

21.2 Merging Characters with the Gray Plane

The gray plane is the mechanism for adding pieces of characters into characters being built. You do this in two steps:

1. Put a character or a piece of a character into the gray plane and position it. You use the [Move Gray] command to reposition a drawing in the gray plane. It leaves the black plane and the character box unaffected; it moves bits within the gray plane only. When you use it, you are asked in the prompt pane for two points, which you indicate by clicking left on them in the drawing pane. These points indicate where *from* and where *to* move the data in the gray plane. FED temporarily grays (in a distinguishable gray) the points you select so that you can see them, and then moves all the data in the gray plane so that the first point is moved to the second. Usually, rather than clicking random points, you should click a specific point in the gray drawing and the point in the black drawing with which you wish the gray point to coincide. You might also think of these points as a point in the gray plane and a point in the black plane to which the point in the gray plane is to be made to coincide.
2. Merge it into the black plane. The [Add in Gray] command merges the gray plane *into* the black plane. Normally, you use [Add in Gray]. This turns on (makes black) each point in the black plane that is "over" a turned-on (gray) point in the gray plane, and leaves the gray plane as it was. Thus, the points that were gray now all appear in dark gray, indicating they are on in both planes. Using [Add in Gray (M)] is similar, but clears the gray plane afterwards.

You can also merge the gray plane into the black plane by other logical operations than the default Inclusive Or: using [Add in Gray (R)] pops up a menu of logical combination operators. ANDCA (turn off all black points corresponding to "on"

points in the black plane, that is, punch a hole in the black plane as indicated by the gray plane) and XOR (flip all points in the black plane that are on in the gray plane) are offered, as well as the default value, IOR.

22. Saving Characters and Pieces of Characters in Registers

FED's gray plane allows you to edit one character or piece of a character. You can also save characters and pieces in *registers*. The register pane shows the contents of registers that can hold characters and pieces of characters for reuse.

22.1 Saving a Drawing Into a Register

You save a drawing (in the black plane, after editing) by clicking left on one of the empty registers (little boxes) in the register pane. Do *not* use the first (upper left-hand) one. Clicking left on an empty register (one that looks blank) saves the current black drawing in that register. Registers are mouse-sensitive, and grow a thick border when you move the mouse over them. Click on an empty register, and the drawing in the black plane appears in that register, in the register pane, and remains there. FED makes every effort to show you a visible piece of that character, so that you know it is there.

22.2 Retrieving the Contents of a Register

To retrieve a register, click left on it, and the contents of the register are transferred into the black plane. If you click on a register that has a drawing in it, that drawing goes into the black plane. If it does not have a drawing in it, the black plane goes into it. Thus, clicking left on registers is usually the only dealing you have with them.

22.3 Retrieving the Black Plane While Manipulating Registers

You might click on a different register than the one you intended. Or perhaps a register is not really empty, but has a peculiar drawing in it that has a gigantic empty middle. In either of these cases, you might lose the very work in the black plane that you were trying to save. Thus, FED always copies the current black plane into the upper left-hand register when fetching the contents of a register, in case you made a mistake. You can then click on the upper left-hand register to retrieve its contents.

Drawings saved in registers are saved as bits; the orientation and size of the character box are not saved.

It is possible to save the gray plane into a register, or fetch a register into the gray plane. It is also possible to store into a nonempty register from either plane. If you want to do any of these operations, click right on a register, and a menu of possible operations pops up.

23. Transformations

Although drawing with the mouse is the most common way to create characters and pieces of characters, FED can provide a good deal of automatic drawing help, such as drawing lines and curves and performing transformations on the character being edited. As is true of drawing with the mouse, all of these operations are applicable only to the black plane. If you want to perform them on the gray plane, swap planes, perform them, and swap back.

23.1 Clearing the Drawing

The simplest operation on a drawing is getting rid of it; [Erase All] clears the entire (black) drawing. The gray drawing, if any, is left intact. You are queried to make sure you really want to clear the entire drawing. This function is also accessible via the keyboard command E.

23.2 Rotating Drawings

FED can rotate characters 90 degrees right or left, or 180 degrees. Rotations are performed about the center of the square whose top, right, and left edges are the top, right and left edges of the character box, and thus, whose bottom must be, and is, a distance below the top of the character box equal to the character width.

<i>Rotation</i>	<i>Mouse command</i>
90 degrees right	[Rotate (R)]
90 degrees left	[Rotate]
180 degrees	[Rotate (M)]

Note that rotating the drawing 180 degrees is not the same as turning it upside down.

23.3 Reflecting Drawings

FED can reflect drawings about any of four lines. Using [Reflect] pops up a menu of the four lines about which to rotate the drawing. Those lines all pass through the "center" of the character box, the point halfway between the left and right edges and halfway between the top and the *bottom line*, *not* the baseline.

The screenshot shows a text editor window with a grid background. A large character 'G' is displayed in the center. To the right of the grid is a menu with various options. Below the menu are font parameters and a character selection table. At the bottom, there is a status bar with system information and a command prompt.

Set Points	Clear Points	Flip Points
Configure	Grid Size	Center View
Move View	Draw Line	Draw Spline
Erase All	Stretch	Rotate
Reflect	Move Black	
Gray Char	Clear Gray	Swap Gray
Move Gray	Add in Gray	
Edit Font	List Fonts	Save Char
Rename Char	Show Font	Set Sample
Read File	Write File	EXIT
HELP		

Font: LWFIX10
 Character: G (107)
 Width: 22

Select Character to edit

•	↓	α	β	^	~	ε	π	λ	γ	δ	†	±
•	∞	∂	c	∫	n	u	Y	∃	•	z	+	+
#	◊	≤	≥	=	v	!	"	/	#	\$	1	%
	()	x	+	,	-	.	<	I	>	K	?
4	5	6	7	8	9	:	; H	<	I	>	K	?
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
[\]	^	~	-	_	a	b	c	d	e	f
h	i	j	k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z	{		}	~	~	~	~

Font parameters (decimal)
 Blinker Height: 42
 Blinker Width: 22
 Base Line: 29
 Character Height: 41

Registers

Rotate the character: L: 90 left, R: 180, R: 90 right

[Thu 17 Jul 3:01:59] whit CL-USER: User Input * GLL:Michl:28091546.rdata 29 65888

Figure 4. [Rotate (R)]

The screenshot shows a font editor window. On the left, a character 'G' is displayed on a grid. On the right, a menu is open with 'Rotate' highlighted. Below the menu is a character set and font parameters.

Set Points	Clear Points	Flip Points
Configure	Grid Size	Center View
Move View	Draw Line	Draw Spline
Erase All	Stretch	Rotate
Reflect	Move Black	
Gray Char	Clear Gray	Swap Gray
Move Gray	Add in Gray	
Edit Font	List Fonts	Save Char
Rename Char	Show Font	Set Sample
Read File	Write File	EXIT
HELP		

Font: LWFIX10
 Character: G (107)
 Width: 22

Select Character to edit

•	α	β	^	~	ε	n	λ	γ	δ	†	±
•	∞	∂	∫	∞	∞	∞	∞	∞	∞	∞	∞
#	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
'	()	*	+	-	.	/	0	1	2	3
4	5	6	7	8	9	:	<	=	>	?	•
A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y
[\]	^	~	ı	a	b	c	d	e	f
h	i	j	k	l	m	n	o	p	q	r	s
u	v	w	x	y	z	{		}	~		

Font parameters (decimal)
 Blinker Height: 42
 Blinker Width: 22
 Base Line: 29
 Character Height: 41

Registers

Rotate the character: L: 90 left, R: 180, R: 90 right

[Thu 17 Jul 3:02:32] whit CL-USER: User Input • GLL:Michlen:20001546.rdata 327 73840

Figure 5. [Rotate (M)]

These lines are the horizontal and vertical lines through the center point, the *X Axis* ($()$) and the *Y Axis* ($-$), and the 45-degree diagonals, the line $X=Y$ ($/$), and the line $X=-Y$ (\backslash), through it.

Reflection is subtle; it is very different than rotation. Imagine the drawing as made of sheet metal, lying on the plane. Rotation moves the character around in the plane, turning it, but never lifting it off the plane. Reflection picks it up, and puts it *back, face down* on the plane. The effects of diagonal reflections are subtle. The best way to understand these commands is to edit an asymmetrical but simple character (the one of choice is F) in a straightforward font (for example, HL12B), and try these various reflections upon it, as well as the rotations.

23.4 Moving the Drawing

You can move the drawing around with [Move Black]. [Move Black] moves the drawing with respect to the character box, the drawing pane itself, and the gray plane. [Move Black] prompts for two points, a point in the black plane and a point to which to move it. The whole black drawing moves along with it as well.

23.5 Drawing Lines and Curves

FED can draw approximate lines and curves in the drawing. Rather than drawing actual lines and curves on the drawing, FED manipulates squares *along* the line or curve desired. Thus, if you ask to draw a line that is not straight up, down, or across, FED approximates as well as it can.

To draw a line, use [Draw Line], and select two points between which to draw a line. As with all commands in which FED prompts for points, the points are temporarily grayed when you click on them, to verify your choices. The line is drawn in the current draw mode, which means it clears a line if appropriate, or even flips all the points along one (which is hardly ever appropriate).

To draw a curve, use [Draw Spline]. Then click left on all the points through which the curve is to pass. When you are done, use [Draw Spline (R)]. The spline-drawing package is called to compute the points of an unconstrained cubic spline through these points, and the approximate curve is drawn in the current draw mode. See the section "Drawing Splines on Windows" in *Programming the User Interface, Volume B*.

G

Select with mouse the point on the BLACK PLANE to move that to...

Set Points	Clear Points	Flip Points
Configure	Grid Size	Center View
Move View	Draw Line	Draw Spline
Erase All	Stretch	Rotate
Reflect	Move Black	
Gray Char	Clear Gray	Swap Gray
Move Gray	Add in Gray	
Edit Font	List Fonts	Save Char
Rename Char	Show Font	Set Sample
Read File	Write File	EXIT
HELP		

Font: LWFIX10
 Character: G (107)
 Width: 22

Select Character to edit	
• ↓ α β γ δ ε η λ γ δ t ±	• ω 3 c 3 n u ! 3 # 0 1 2 3 8 3
# () * + 8 9 : ; / < = > ?	4 5 6 7 8 9 : ; / < = > ?
A B C D E F G H I J K L M Z	A B C D E F G H I J K L M Z
N O P Q R S T U V W X Y Z	N O P Q R S T U V W X Y Z
[\] ^ _ ` a b c d e f g h	[\] ^ _ ` a b c d e f g h
i j k l m n o p q r s t	i j k l m n o p q r s t
u v w x y z { } ~	u v w x y z { } ~

Font parameters (decimal)
 Blinker Height: 42
 Blinker Width: 22
 Base Line: 29
 Character Height: 41

Registers

! Draw dots, R Change mode (Draw/Erase/Flip), M Move edges of char box, R2: System menu

[Thu 17 Jul 3:09:53] whit CL-USER: User Input * QLL:Michloep:90901546.rdata R32 189712

Figure 8. Moving the Drawing with [Move Black]

23.6 Stretching and Contracting

FED can stretch or contract drawings. This is not the same as growing or shrinking them. Stretching means inserting duplicate rows or columns at a given point of the drawing, and contracting means removing rows or columns. Growing and shrinking, in general, mean scaling the whole drawing up or down. The latter is done with the options to [Gray Char]. See the section "Getting Things Into the FED Gray Plane", page 303.

The relative orientation of the first and second points clicked on specifies whether you want to stretch or to contract.

23.6.1 Stretching a Drawing Horizontally

Stretching a drawing horizontally means making some number of copies of a column of squares to the right of that column. To stretch a character horizontally, use [Stretch], and then click left on any square in the column to be "stretched". Then click left on any square in the column to the right of that to which that column is to be stretched (that is, the last column to be a duplicate of the column being stretched). The entire drawing is stretched, with the required number of copies of the duplicated column inserted.

23.6.2 Contracting a Drawing Horizontally

Contracting a drawing horizontally means eliminating some number of columns of squares. To shrink a character horizontally, use [Stretch]. Then click left on any square in the rightmost column not to be eliminated, at the right edge of the columns to go, and then on the leftmost column to be eliminated. You should think of this as clicking on a column to move, and where to move it to.

23.6.3 Stretching a Drawing Vertically

Stretching a drawing vertically means making some number of copies of a row of squares below that row. To stretch a character vertically, use [Stretch (M)], and then click left on any square in the row to be "stretched". Then click left on any square in the row below that to which that row is to be stretched (that is, the last row to be a duplicate of the row being stretched). The entire drawing is stretched, with the required number of copies of the duplicated row inserted.

23.6.4 Contracting a Drawing Vertically

Contracting a drawing vertically means eliminating some number of rows of squares. To shrink a character vertically, use [Stretch (M)]. Then click left on any square in the topmost row not to be eliminated, at the top edge of the rows to

go, and then on the topmost row to be eliminated. You should think of this as clicking on a row to move, and where to move it to.

24. The Sample String

When you edit a font, it is usually convenient to maintain a *sample string*, displayed in the font, so that you can see how the character you are editing looks in the context of other characters next to which it might appear.

FED allows you to set a sample string. The straightforward method of setting it is to select the [Set Sample] menu item: doing so prompts you for the string, which should be short enough to fit in the sample pane (it is clear if it does not, as you only see the end of it). End the string by pressing RETURN. The string is then displayed in the sample pane.

If the sample string contains the character being edited, occurrences of that character are updated whenever any change is made to the drawing. Thus, the occurrences of the character being edited in the sample string reflect the state of the current drawing, not the state of that character stored in the font.

Two other ways to ask FED to prompt you for the sample string are clicking any button on the sample pane itself, and issuing the `U` command from the keyboard. This last is often the most convenient, because you are then going to type the string itself.

25. Adjusting the Display

The commands and facilities described here deal with positioning the drawing display and modifying its visible characteristics. They do not actually change the data in the drawing, but rather, the way it is viewed.

25.1 Positioning the Drawing

Both the black and gray drawings can be thought of as being drawn on an infinite plane. The character box is in the center of that plane. Although [Move Black] and [Move Gray] exist to move the drawings, and the character box can be moved by clicking on it, sometimes you might want to reposition the entire drawing, character box, black drawing, gray drawing, and all. This can also be viewed as repositioning the *view* of the drawing offered by the drawing pane. FED provides several techniques for repositioning the entire drawing.

[Move View] The simplest is [Move View]. [Move View] works just like [Move Gray] and [Move Black]. When you use [Move View], it prompts you for two points, which you indicate by clicking left on squares on the drawing pane. The first point is a point on the drawing; the second is a point in the pane to which to move it. The whole drawing is moved, perhaps simultaneously vertically and horizontally, so that the first point is where the second point had been.

[Center View] Another common need is to recenter the drawing, that is, put the character box back in the middle. This is the way the drawing pane starts out when you begin editing a character. The [Center View] menu item performs this task. Use [Center View] to recenter the drawing. The keyboard H (for Home) command does this too.

Scrolling Another way to reposition the display is to *scroll* it up or down or left or right. In order to scroll the display vertically, a scroll bar is provided at the left of the drawing pane. When you move the cursor to the extreme left edge of the drawing pane and bounce the cursor at that edge, the cursor changes to a double-pointed arrow and the left margin of the drawing pane displays a graph of the vertical portion of the drawing you are looking at. The status line documentation reflects the possible options at this point.

To scroll the drawing horizontally, a scroll bar is provided at the bottom of the drawing pane. When you move the cursor to the extreme bottom edge of the drawing pane and bounce the cursor at that edge, the cursor changes to a double-pointed arrow and the bottom edge of the drawing pane displays a full-grid length graph of what horizontal portion of the drawing you are looking at. The status line documentation reflects the possible options at this point.

25.2 Setting the Box Size in the Drawing Pane

You can set the size of boxes in the drawing pane. Normally, it is 12, meaning each box, corresponding to one pixel of the actual character, is represented by a box 12 pixels wide and high. To set the size of boxes, use [Grid Size]. FED prompts you for the size of a box, in decimal. This size can not be bigger than 64 pixels. If you type a carriage return without typing any number, the default size of 12 pixels is reestablished.

25.3 Setting the Height and Width of the Drawing Pane

You can tell the FED frame to show either a wide drawing pane, as wide as the screen, or a tall drawing pane, almost as tall as the screen. These two *configurations* of the frame are chosen from a pop-up menu that is obtained by using [Configure]. This command can also be used to have FED recompute its configuration, for example, to reshape its registers after you have edited the Space character of a font.

26. Reading and Writing Files

FED can read and write files containing fonts in any of a variety of formats. The most common format is BFD, the standard font format of the Symbolics Lisp Machine. If you are making fonts for use by the Symbolics Lisp Machine display and window system or the LGP-1, this is the only format you should ever have to deal with.

Most of the other formats are for compatibility with other systems and earlier releases of the Symbolics Lisp Machine software. Notable among these formats is PXL format, which is a standard font format with the *TEX* system on UNIX. BFD format is the default for all file reading and writing operations.

26.1 Reading Files

Use [Read File] and type in the file name to read in a font file. The file type defaults from the (canonical) type of the pathname presented as the default. For example, if you type `fix9.bfd`, or just `fix9`, you read a BFD file, whereas if you type `fix9.bin`, you read a BIN file. FED complains if you supply a file type that is not a valid font file type for the machine you are using. Pressing `R` is equivalent to using [Read File].

From outside of FED you can use `Dired` to read in any font file. Enter `Dired`, move point to the line showing the font file, and press `R` (which queues a file to be acted on by a function). Apply the `zl:fed` function to that file to read it in.

When you read in a font via [Read File], it is actually loaded. It becomes part of the Lisp environment, and appears in listings of loaded fonts produced by [List Fonts] as well as by the `Show Font` command and by `Zmacs`. After FED loads the file and looks for the font you specified, you are editing that font.

It is sometimes necessary to read in font files of exotic types, whose file types (as expressed in the name of the file) are not indicative of the format of the font. For instance, you might have renamed a BFD or other file to `myfont.temp`, and now you want to read it in. Since FED cannot determine the font format from this file type, you must specify the font format explicitly. This is done by using [Read File (R)]: FED offers a menu specifying file types. Click on the file type involved: FED then prompts for a pathname and reads the file. FED interprets the file, however, according to the format specified by the menu, not by the file type.

26.2 Writing Files

FED can also write out font files. Files are written from the description of a font residing in the Lisp environment, not from any temporary FED image of the font. Since FED maintains no temporary image of the font, but actually stores edited characters back in the font when you use [Save Char], this is not a problem unless you forget to save your characters.

Use [Write File] to write the font file out. The file type defaults from the (canonical) type of the pathname presented as the default. For example, if you type `newfnt.bfd`, you write a BFD file, whereas if you type `newfnt.bin`, you write a BIN file. FED complains if you supply a file type that is not a valid font file type for the machine you are using. Using [Write File] writes out a BFD file by default from a font description in the Symbolics Lisp Machine virtual memory. The default directory is the system screen fonts directory; the default file name is `font.bfd`, where *font* is the current font being edited. Pressing `⌘` is equivalent to using [Write File].

It is sometimes necessary to write out font files of exotic types, whose file types (as expressed in the name of the file) are not indicative of the format of the font. For instance, you might already have a `sfnt.bfd`, and want to write your file to `sfnt.temp`. Since FED cannot determine the font format from this file type, you must specify the font format explicitly. This is done by using [Write File (R)]: FED offers a menu specifying file types. Click on the file type involved: FED then prompts for a pathname and writes the file. FED writes the file, however, according to the format specified by the menu, not by the file type.

27. Command List

The following is a listing of all FED commands. The first part of this listing describes the commands available via the command menus and the keyboard. When a keyboard character exists duplicating a menu command, it is given in addition after the command name. The second part of this section describes the effect of clicking on various panes and mouse-sensitive areas of the FED frame.

Many of the keyboard commands take *numeric arguments* to specify some number or character. Numeric arguments are entered by typing a decimal number before the command character. The numeric argument is echoed in the prompt window as you enter it.

27.1 Menu and Keyboard Commands

27.1.1 Configuration and Drawing Transformation

[Configure]	Pop up a menu of frame configurations. Two configurations are offered, giving a tall and wide aspect ratio to the drawing pane.
[Grid Size] @	Set the size of boxes in the draw pane. If a numeric argument is given, it is used as the size. @ sets grid size to the default if given no numeric argument, but [Grid Size] prompts.
[Center View] H	Reposition the display in the drawing pane so that the character box is centered in it.
[Move View]	Reposition the display in the drawing pane by prompting for two mouse-specified points: which point to move and to which point to move it.
[Draw Line]	Draw a line in squares in the drawing pane, in the current drawing mode. Prompt for two endpoints, to be specified with the mouse.
[Draw Spline]	Draw a cubic spline in squares in the drawing pane, in the current drawing mode. Prompt for curve points, to be specified by using [Draw Spline]. Using [Draw Spline (R)] ends the curve.
[Erase All] E	Clear all points (black points) in the current drawing.
[Stretch] K	Stretch or contract a character, horizontally or vertically. [Stretch] is horizontally, [Stretch (R)] is vertically. FED

prompts for two points, specifying a row or column to move and to where to move it. From the keyboard, K means horizontal, c-K means vertical. See the section "Stretching and Contracting Drawings in FED", page 316.

- [Rotate] ⊕ Rotate the drawing in the black plane. [Rotate] is 90 degrees to the left, [Rotate (R)] 90 degrees to the right, and [Rotate (M)] 180 degrees.
- [Reflect] ↔ Reflect the drawing in the black plane about a coordinate axis or diagonal line through the center of the character box. A menu pops up, asking which.
- [Move Black] Move the drawing in the black plane. You are prompted for the target and destination points, which you specify by clicking left on the drawing pane.

27.1.2 Gray Plane Menu Items

[Gray Char] G, also M

Place a character into the gray plane. The keyboard commands accept numeric arguments to specify which character. If none is given, or if you use [Gray Char], you are prompted for a character, which you can supply from the keyboard or the Character Select menu. If you use [Gray Char (R)], you are offered a Choose Variable Values choice window to select the character, font, and scaling. For the keyboard commands, CONTROL causes FED to prompt for a font name, and META causes it to prompt for scale factors.

- [Clear Gray] Clear the entire gray plane.
- [Swap Gray] Exchange the drawings in the gray and black planes.
- [Move Gray] Move the drawing in the gray plane. You are prompted for two points, to be specified via the mouse, a point to move and a point to which to move it.
- [Add in Gray] Combine the drawing in the gray plane into the black plane. Using [Add in Gray] inclusive-or's the gray drawing into the black drawing. Using [Add in Gray (M)] inclusive-or's the gray drawing into the black drawing, and clears the gray drawing. [Add in Gray (R)] pops up a menu of other combination modes.

27.1.3 Outside World Interface Menu Items

- [Edit Font] F Pick a font to edit. You are prompted for the font name. Use [Edit Font (M)] to copy an existing font as the first step of making a new font.
- [List Fonts] [List Fonts] lists all of the loaded fonts. [List Fonts (R)] lists all of the loaded fonts and fonts on the file computer. The display is mouse-sensitive; clicking left on any item begins editing that font.
- [Save Char] S Store the character being edited back into the font in the Lisp environment. It is stored as the character that the status pane indicates it to be.
- [Rename Char] c-C Rename the current character; make it seem as though you are now editing a different character, but retain the drawing. You are prompted for the character, which you can supply from either the keyboard or the Character Select menu. The keyboard command accepts a numeric argument to specify the character.
- [Show Font] D Display all characters in the font being edited. The display is mouse-sensitive, and clicking left on a character begins editing that character.
- [Set Sample] V Prompt for the sample string to be displayed in the font being edited in the sample pane, and set it.
- [Read File] R Read in a file of font definitions. Prompts for a pathname. [Read File] computes the font file type from the file type of the pathname given. The default is always BFD. [Read File (R)] pops up a menu that offers the file types: BFD, KST, BIN, AC, AL, PXL, or Any. The file specified by the pathname given will be interpreted according to that format, regardless of file type.
- [Write File] W Writes a file of font definitions. Prompts for a pathname. [Write File] computes the font file type from the file type of the pathname given. The default is always BFD. [Write File (R)] pops up a menu that offers the file types: BFD, KST, BIN, AC, AL, PXL, or Any. The file specified by the pathname given will be written in that format, regardless of file type.
- [EXIT] Q Bury the Font Editor, and return to whatever you were doing when you last invoked FED.
- [HELP] HELP or ? Display a long message giving documentation of FED.

27.1.4 Evaluating Forms From FED

FED uses the ESCAPE key to evaluate a Lisp form.

27.2 Keyboard-only Commands

The following commands are accessible only from the keyboard. They are mainly concerned with the nonmouse cursor, or general interaction with the subsystem.

<code>\</code>	Turn the nonmouse cursor on, and move it one position up the screen. A numeric argument tells to move it other than one position. <code>c-\</code> and <code>m-\</code> mean 2 and 4 positions, respectively, and <code>c-m-\</code> means 8.
<code>/</code>	Same as <code>\</code> , but moves the nonmouse cursor down.
<code>[</code>	Same as <code>\</code> , but moves the nonmouse cursor left.
<code>]</code>	Same as <code>\</code> , but moves the nonmouse cursor right.
<code>.</code>	When the nonmouse cursor is on, complement the black square under it.
REFRESH	Redraw the drawing pane. Useful in case of perceived problems.
<code>c-REFRESH</code>	Clear the screen and refresh all panes in the FED frame.
ABORT	Abort any command while it is prompting, waiting for either mouse or keyboard input.
<code>C</code>	Begin editing a character: prompt for the character, and begin editing it. Normally, you simply select a character from the Character Select menu or the [Show Font] display. <code>C</code> accepts a character specification as a numeric argument.

27.3 Mouse Sensitivities

This section describes the result of clicking the mouse on various portions of the FED frame other than the command menus.

27.3.1 The Drawing Pane

Click left	Draw a black square in the current draw mode, which is shown by the Draw Mode menu. It continues drawing as the mouse is moved as long as the left button is held down. Pressing
------------	--

CONTROL while drawing means temporarily go into [Clear Points] mode (META means [Set Points] mode); neither changes any points.

- Click middle Change the draw mode, cycling through the three possible draw modes.
- Click right Only meaningful when the mouse is over a boundary of the character box. "Pick it up" and begin moving it as the mouse is moved, as the right button is held down.

The drawing pane has a scroll bar at its left edge.

27.3.2 The Draw Mode Menu

Clicking any button on one of the draw modes selects that draw mode until it is next changed by clicking on this menu, or clicking middle on the drawing pane.

27.3.3 The Sample Pane

Clicking any button on the sample pane prompts for a new sample string.

27.3.4 The Character Select Pane

Clicking left on any character in the character select pane begins editing it. The character select pane can also be used to answer any command that is prompting for a character.

27.3.5 The Font Parameters Menu

Clicking left on any item in the Font Parameters menu opens it for editing. You are expected to type a new decimal number. As soon as you press RETURN, the altered parameter is stored in the font in the Lisp environment.

27.3.6 The Register Pane

- Click left On an empty register, store the current black plane drawing in that register. On a nonempty register, retrieve the drawing in it into the black plane, and store the current black plane drawing into the upper-leftmost register.
- Click right Pop up a menu allowing the register you clicked on to be loaded from either plane (regardless of whether or not it is empty) or retrieved to either plane.

27.3.7 The List Fonts and Show Font Displays

These displays are mouse-sensitive. Clicking left on a font in the [List Fonts] display begins editing it; clicking left on a character in the [Show Font] display begins editing that character.

Index

!	!	!
	! Dired command 172	
Exclamation point	(!) line continuation indicator 25, 64	
#	#	#
	@ # text formatting command 39	
\$	\$	\$
	\$ Dired command 171	
)))
	c-X) Zmacs command 257	
*	*	*
*Function-Specs-to-Edit-n	* buffer 13	
	Function-Specs-to-Edit-n buffer 13	
	@ * text formatting command 39	
+	+	+
	+ flag in Zmacs 131	
,	,	,
	, Dired command 168	
.	.	.
	. Font Editor drawing command 300	
	@ . text formatting command 39	
/	/	/
	/ Font Editor command 330	
1	1	1
	c-X 1 Zmacs command 147	

2	2 c-X 2 Zmacs command 146	2
3	3 c-X 3 Zmacs command 146	3
4	4 c-X 4 Zmacs command 146	4
8	8 c-X 8 Zmacs command 146	8
;	; c-X ; Zmacs command 227	;
=	= = Dired command 169 c-X = Zmacs command 54 @ = text formatting command 39	=
>	> @ > text formatting command 39	>
?	? ? Dired command 167 ? Font Editor command 329 HELP ? Zmacs command 14	?
@	@ @ Font Editor command 327 @ text formatting command 39 c-m- @ Zmacs command 106 m- @ Zmacs command 106 @# text formatting command 39 @* text formatting command 39 @. text formatting command 39 @= text formatting command 39 @> text formatting command 39 @blankspace text formatting command 39 @b text environment 36 @caption text formatting command 39 @c text environment 36 @foot text formatting command 39 @g text environment 36 @i text environment 36 @note text formatting command 39	@

@p text environment 36
 @r text environment 36
 @tabclear text formatting command 39
 @tabdivide text formatting command 39
 @tabset text formatting command 39
 @t text environment 36
 @\ text formatting command 39
 @^ text formatting command 39

A**A****A**

Example of a Search String for HELP A 53
 A Dired command 172
 A Zmacs command 141
 A Zmacs command 14, 53
 c-X c-A Add Mode Word Abbrev 200
 c-X plus-SIGN Add Global Word Abbrev 200
 Make Word Abbrev 202
 Word abbrev 198
 Read Word Abbrev File 202
 Write Word Abbrev File 203
 Word Abbreviation Commands 200
 Using Word Abbreviations 198
 Word Abbreviations 197
 Word Abbrev Mode 203
 Edit Word Abbrevs 201
 Insert Word Abbrevs 201
 Kill All Word Abbrevs 201
 List Some Word Abbrevs 202
 List Word Abbrevs 202
 Word abbrevs 198
 Dired Abort 167
 ABORT Dired command 167
 ABORT Font Editor command 330
 ABORT Zmacs command 48
 Abort At Top Level 48
 Abort Patch (m-X) 248
 About Buffers and Regions 61
 about file attribute lists 156
 About Flavors 281
 About Lisp 280
 about patch 247
 About the State of Buffers 278
 About the State of Zmacs 279
 About Zmacs Commands 51
 About Zmacs Commands 53
 About Zmacs Commands 51
 About Zmacs Commands with HELP 51
 About Zmacs Variables 269
 Accept command 181
 Accept Once command 181
 Accidental deletion 49
 Active patches 242, 246
 Add Global Word Abbrev 200
 [Add in Gray] Font Editor menu item 304, 328
 Adding Site-specific Speller Dictionaries 189
 Adding User-specific Speller Dictionaries 188
 Add Mode Word Abbrev 200
 Add Patch Changed Definitions (m-X) 245
 Add Patch Changed Definitions of Buffer (m-X) 245
 Add Patch (m-X) 244

- Assign key bindings 267
- Associating a File with a Buffer 139
- Association of buffers with files 32
- Association of files with buffers 32
- Atom Query Replace 121
- Backspace Attribute 159
- Backspace file attribute 159
- Base attribute 157, 160
- Base file attribute 160
- Baseline Font Attribute 286
- Character Height Font Attribute 285
- Character Width Font Attribute 286
- Chars-exist-table Font Attribute 287
- Fixed-width Font Attribute 287
- Fonts Attribute 160
- Left Kern Font Attribute 286
- Lowercase Attribute 160
- Lowercase file attribute 160
- Nofill Attribute 160
- Nofill file attribute 160
- Patch-File Attribute 161
- Patch-File file attribute 161
- Syntax attribute 157
- Tab-Width Attribute 161
- Tab-Width file attribute 161
- Unknown attribute 156
- Vsp Attribute 161
- Vsp file attribute 161
- File Attribute Checking 156
- Buffer and File Attribute Descriptions 159
- Attribute list 155
- Describe Attribute List 168
- Reparse Attribute List (m-X) Zmacs command 155
- Update Attribute List (m-X) Zmacs command 156
- Update Attribute List Query 159
- Attribute lists 156
- Warnings about file attribute lists 156
- Attribute-Manipulating Commands 155
- Example of Attribute-Manipulating Commands 156
- Attributes 155
- Blinker Width and Blinker Height Font Attributes 287
- Buffer attributes 155
- Character attributes 285
- File attributes 155
- Font attributes 285
- Other Set Commands for File and Buffer Attributes 159
- Set commands for file and buffer attributes 159
- Setting Buffer Attributes 159
- Viewing File Attributes in Dired 168
- Buffer and File Attributes in Zmacs 155
- Attributes of TV Fonts 285
- Auto Fill in Text Mode 274
- Init File Form: Auto Fill Minor Mode 253
- Example of Filling Text with Auto Fill Mode 160, 254
- Overview of Locating and Replacing Strings Automatically 118
- Locating and Replacing Strings Automatically in Zmacs 118
- Automatic drawing help 309

- Inserting Blank Line in Zmacs 221
- Delete Blank Lines 30
- @ blankspace text formatting command 39
- Blinker Width and Blinker Height* 287
- Blinker Height* Font Attributes 287
- Blinker width* 287
- Blinker Width and Blinker Height* Font Attributes 287
- Boldface text environment 36
- Bolio Mode 177
- Bottom 146
- Bottom Edge of the FED Character Box 301
- Box 302
- Box 301
- box 291, 301, 302
- Box 302
- FED Character Box 291
- Left and Right Edges of the FED Character Box 301
- Top Edge of the FED Character Box 301
- Using the mouse on the character box 302
- Viewing and Altering a Character in the FED Character Box 301
- What the Lines Mean in the FED Character Box 301
- Size of boxes in the drawing pane 324, 327
- Setting the Box Size in the FED Drawing Pane 324
- Breaking a line 24
- *Function-Specs-to-Edit-n* buffer 13
- Appending a Region to a Buffer 141
- Append To Buffer 141
- Associating a File with a Buffer 139
- Beginning/End of Buffer 81
- Creating a Buffer 32, 33
- Creating a Fundamental Mode Buffer 139
- Current buffer 137
- Current Zmacs Buffer 132
- Editor Window's Buffer 18
- Encrypting and Decrypting the Buffer 137
- Evaluate and Replace Into Buffer 120
- Execute Command Into Buffer 136
- Format Buffer 42
- Hardcopying the Buffer 136
- Inserting a Buffer Into Another Buffer 141
- Inserting a File Into a Buffer 141
- Inserting Command Output Into the Buffer 136
- Inserting output into the buffer 136
- Insert text from register into buffer 105
- m-X Spell Buffer 180, 185
- Marking a Region From Here to Beginning of Buffer 107
- Marking a Region From Here to End of Buffer 107
- Mode Line's Buffer 21
- Motion with Respect to the Whole Buffer 81
- Moving to end of buffer 81
- Reading a File Into a New Buffer 137
- Reading a File Into an Existing Buffer 137
- Renaming the Buffer 136
- Re-reading a File Into the Buffer 138
- Select Buffer 32, 133
- Select Default Previous Buffer 133
- Selected buffer 132
- Select Previous Buffer 133
- Showing a Buffer 135
- The Editor Window and the Buffer 64
- Undo all changes to buffer 138

- View Buffer 135
- Buffer and File Attribute Descriptions 159
- Buffer and File Attributes in Zmacs 155
- Zmacs Buffer and File Names 130
- Buffer attributes 155
- Other Set Commands for File and Buffer Attributes 159
- Set commands for file and buffer attributes 159
- Setting Buffer Attributes 159
- Zmacs Buffer Commands 133
- Writing the Buffer Contents to a File 138
- Saving the Buffer Contents to the File 138
- Buffer Contents with c-X c-F 33
- Buffer Flags for Existing Files 130
- Buffer Flags for New Files 131
- Buffer History 132
- Zmacs Buffer History 132
- Inserting a Buffer Into Another Buffer 141
- Init File Form: Ordering Buffer Lists 272
- Add Patch Changed Definitions of Buffer (m-X) 245
- Evaluate Buffer (m-X) Zmacs command 230
- Evaluate and Replace Into Buffer (m-X) Zmacs command 230
- Evaluate Changed Definitions of Buffer (m-X) Zmacs command 230
- Evaluate Into Buffer (m-X) Zmacs command 230
- Execute Command Into Buffer (m-X) Zmacs command 136
- Format Buffer (m-X) Zmacs command 35, 42
- Hardcopy Buffer (m-X) Zmacs command 136
- Insert Buffer (m-X) Zmacs command 141
- List Changed Definitions of Buffer (m-X) Zmacs command 239
- Rename Buffer (m-X) Zmacs command 136
- Revert Buffer (m-X) Zmacs command 138
- Show Buffer (m-X) Zmacs command 135
- Buffer pointers 100
- Buffers 32
- Association of files with buffers 32
- Changing Buffers 133
- Commands to Mark Regions by Buffers 107
- Destroying Buffers 139
- Editing Buffers 134
- Example of Listing Buffers 134
- File buffers 139
- List Buffers 134
- Listing Buffers 133
- Multiple buffers 130
- Possibility Buffers 126
- Reverting buffers 138, 139
- Saving Buffers 136
- Selecting, Listing, and Examining Zmacs Buffers 132
- Support Buffers 126
- Zmacs Commands for Finding Out About the State of Buffers 278
- Creating and Saving Buffers and Files 32
- Description of Creating and Saving Buffers and Files 32
- Summary of Creating and Saving Buffers and Files 32
- Manipulating Buffers and Files in Zmacs 129
- Overview of Working with Buffers and Files in Zmacs 130
- Working with Buffers and Files in Zmacs 130
- Getting Information About Buffers and Regions 61
- Init File Form: Putting Buffers Into Current Package 272
- Changing Case of Buffers in Zmacs 214
- Comparing Files and Buffers in Zmacs 142
- Edit Buffers (m-X) Zmacs command 134
- Init File Form: Edit Buffers on c-X c-B 274

Init File Form: Edit Buffers on m-X 275
 Mode Line's *Buffer-status* 22
 Association of buffers with files 32
 List Buffers Zmacs command 131
 Built-in Customization Using Zmacs Minor
 Modes 253

C

C

C

HELP C 51
 SELECT C 44
 Selecting a FED Character with the C Command 298
 C Dired command 169
 C Font Editor command 298, 330
 HELP C Zmacs command 14, 52
 c-% Zmacs command 118
 c-/ completion command 15
 c-0 c-m-Y 86
 c-0 c-m-Y yank command 15
 c-0 c-Y 85
 c-; Zmacs command 226
 c-X c-; Zmacs command 228
 c-= Zmacs command 54
 HELP or c-? 15
 c-? completion command 15
 c-X c-A Add Mode Word Abbrev 200
 c-A Zmacs command 28, 74, 95
 Init File Form: Edit Buffers on c-X c-B 274
 c-B Zmacs command 28, 72
 c-B Zmacs command 131, 134
 c-C Font Editor command 329
 c-D Dired command 170
 c-D Zmacs command 30, 49, 90
 c-X c-D Zmacs command 149
 c-E Zmacs command 28, 74, 95
 Buffer Contents with c-X c-F 33
 c-F Zmacs command 28, 72
 c-X c-F Zmacs command 33, 34
 c-G Zmacs command 48
 c-HELP V Zmacs command 269
 c-X c-I Zmacs command 220
 c-J Change Style Character 210
 c-X c-J Change Style Region 210
 c-K Dired command 170
 c-K Zmacs command 30, 95
 c-L Zmacs command 65
 c-X c-L Zmacs command 214
 c-m- (Zmacs command 77
 c-m-) Zmacs command 77
 c-m-; Zmacs command 226
 c-m-? V Zmacs command 269
 c-m-@ Zmacs command 106
 c-m-A Zmacs command 78
 c-m-B Zmacs command 76
 c-m-D Zmacs command 77
 c-m-E Zmacs command 78
 c-m-F Zmacs command 76
 c-m-H Zmacs command 106
 c-m-J Change Typein Style 211
 c-m-K Zmacs command 30, 93
 Init File Form: c-m-L on the SQUARE Key 274

- c-m-L Zmacs command 133
- c-X c-m-L Zmacs command 133
- c-m-N Zmacs command 76
- c-m-O Zmacs command 221
- c-m-P Zmacs command 76
- c-m-Q Zmacs command 219
- c-m-R Zmacs command 66
- c-m-RUBOUT Zmacs command 30, 93
- c-m-sh-E Zmacs command 230
- c-m-SPACE Zmacs command 102
- c-X c-m-SPACE Zmacs command 102
- c-m-T Zmacs command 93
- c-m-U Zmacs command 77
- c-m-V Zmacs command 147
- c-m-X 7
- Kill Backward Up List (c-m-X) Zmacs command 93
- c-Ø c-m-Y 86
- c-m-Y yank command 15
- c-Ø c-m-Y yank command 15
- c-m-Z Zmacs command 230
- c-m-[Zmacs command 78
- c-m-\ Zmacs command 219
- c-m-] Zmacs command 78
- c-m-~ Zmacs command 220
- c-N Dired command 168
- c-N Zmacs command 28, 74, 95, 257
- c-X c-N Zmacs command 75
- c-Ø c-Y yank command 15
- c-Ø Zmacs command 221
- c-X c-Ø Zmacs command 30, 221
- c-P Dired command 168
- c-P Zmacs command 28, 74, 95
- c-X c-P Zmacs command 107
- c-R Font Editor command 329
- c-REFRESH Font Editor command 330
- c-X c-S Zmacs command 34, 138
- c-sh-A Zmacs command 55
- c-sh-C Zmacs command 109
- c-sh-D Zmacs command 55
- c-sh-E Zmacs command 230
- c-sh-V Zmacs command 55
- c-sh-Y string-matching yank command 87
- c-SPACE Zmacs command 102
- c-T Zmacs command 90
- c-X c-T Zmacs command 96
- c-U Zmacs command 26
- c-X c-U Zmacs command 214
- c-V Zmacs command 28, 65
- c-X c-V Zmacs command 137
- c-W Font Editor command 329
- c-W Zmacs command 30, 109
- c-X c-W Zmacs command 34, 138
- c-X) Zmacs command 257
- c-X 1 Zmacs command 147
- c-X 2 Zmacs command 146
- c-X 3 Zmacs command 146
- c-X 4 Zmacs command 146
- c-X 8 Zmacs command 146
- c-X ; Zmacs command 227
- c-X = Zmacs command 54
- c-X A Zmacs command 141

- c-X B Zmacs command 32, 33, 133
- c-X c-; Zmacs command 228
- c-X c-A Add Mode Word Abbrev 200
- c-X c-B 274
- c-X c-B Zmacs command 131, 134
- c-X c-D Zmacs command 149
- c-X c-F 33
- c-X c-F Zmacs command 33, 34
- c-X c-I Zmacs command 220
- c-X c-J Change Style Region 210
- c-X c-L Zmacs command 214
- c-X c-m-L Zmacs command 133
- c-X c-m-SPACE Zmacs command 102
- c-X c-N Zmacs command 75
- c-X c-O Zmacs command 30, 221
- c-X c-P Zmacs command 107
- c-X c-S Zmacs command 34, 138
- c-X c-T Zmacs command 96
- c-X c-U Zmacs command 214
- c-X c-V Zmacs command 137
- c-X c-W Zmacs command 34, 138
- c-X c-X Zmacs command 103
- c-X D Zmacs command 163
- c-X E Zmacs command 258
- c-X F Zmacs command 32, 254
- c-X L Zmacs command 55
- c-X O Zmacs command 147
- c-X plus-SIGN Add Global Word Abbrev 200
- c-X Q Zmacs command 260
- c-X RUBOUT Zmacs command 30, 97
- c-X S Zmacs command 32
- c-X T Zmacs command 109
- c-X U Unexpand Last Word 203
- c-X V Zmacs command 135
- c-X W Zmacs command 32
- c-X Zmacs command 257
- c-X [Zmacs command 28, 80
- c-X] Zmacs command 28, 80
- c-X ^ Zmacs command 146
- c-Ø
- c-Y 85
- c-Y yank command 15, 87
- c-Ø
- c-Y yank command 15
- c-Z 45
- C-Ø c-m-sh-Y 86
- C-Ø c-sh-Y 85
- The Zmacs Edit Callers Commands 239
- Multiple Edit Callers Intersection 240
- Multiple List Callers Intersection 241
- Example of Calling the Last Keyboard Macro 258
- Calling the Last Keyboard Macro 258
- Cancel last command 48
- Cancel response 48
- Canonical types 152
- @ caption text formatting command 39
- Carriage return 7
- Changing Case and Indentation in Zmacs 213
- Changing Case in Zmacs 214
- Overview of Changing Case in Zmacs 214
- Changing Case of Buffers in Zmacs 214
- Changing Case of Regions in Zmacs 214

- Changing
 - Case of Words in Zmacs 214
 - Centering the Current Line in Zmacs 219
 - Centering the Drawing in FED 323
 - Center text environment 36
 - [Center View] Font Editor menu item 323, 327
- Add Patch
 - Changed Definitions (m-X) 245
- Evaluate
 - Changed Definitions (m-X) Zmacs command 230
- Add Patch
 - Changed Definitions of Buffer (m-X) 245
- Evaluate
 - Changed Definitions of Buffer (m-X) Zmacs command 230
- List
 - Changed Definitions of Buffer (m-X) Zmacs command 239
- Compile
 - Changed Definitions of Tag Table 231
 - Change File Properties (m-X) Zmacs command 150
 - Change One Style Region 210
- Undo all
 - changes to buffer 138
- c-J
 - Change Style Character 210
- c-X c-J
 - Change Style Region 210
- m-J
 - Change Style Word 211
- c-m-J
 - Change Typein Style 211
- Changing Buffers 133
- Changing Case and Indentation in Zmacs 213
- Changing Case in Zmacs 214
- Overview of
 - Changing Case in Zmacs 214
 - Changing Case of Buffers in Zmacs 214
 - Changing Case of Regions in Zmacs 214
 - Changing Case of Words in Zmacs 214
 - Changing File Properties in Dired 168
 - Changing the Properties of a File 150
 - Changing Window Size 146
- Getting Things Into Gray with [Gray Char] 303
 - Backward
 - Character 72
 - c-J Change Style
 - Character 210
 - Contracting a
 - character 327
 - Deleting the Current
 - Character 90
 - Deleting the Last
 - Character 90
 - Drawing a
 - character 291
 - Editing a
 - character 291
 - Forward
 - Character 72
 - Motion by
 - Character 71
 - RUBOUT Zmacs
 - character 90
- Selecting a FED Character by Creating a New
 - Character 298
- Stretching a
 - character 327
 - Character attributes 285
 - Character box 291, 301, 302
- Altering the FED
 - Character Box 302
- Bottom Edge of the FED
 - Character Box 301
- Character Height of the FED
 - Character Box 302
- FED
 - Character Box 291
- Left and Right Edges of the FED
 - Character Box 301
- Top Edge of the FED
 - Character Box 301
- Using the mouse on the
 - character box 302
- Viewing and Altering a Character in the FED
 - Character Box 301
- What the Lines Mean in the FED
 - Character Box 301
- Selecting a FED
 - Character by Creating a New Character 298
- Selecting a FED
 - Character by Renaming Characters 298
- Prefix
 - character commands 7
- Selecting a FED
 - Character From the Character Select Menu 298
- Selecting a FED
 - Character From the [Show Font] Display 298
 - Character height 285, 291, 302
 - Character Height Font Attribute 285

Current meaning of mouse	clicks 67
C-0	c-m-sh-Y 86
Finding source	code 224
Init File Form: White Space in Lisp	Code 274
Commenting Lisp	Code in Zmacs 226
Introduction to Locating Source	Code in Zmacs 235
Overview of Commenting Lisp	Code in Zmacs 226
Editing the source	code of a function 12
Locating Source	Code to Edit in Zmacs 235
Set Fill	Column 254
Set Goal	Column 75
Goal	Column and the Motion Commands 74
Init File Form: Setting Goal	Column for Real Line Commands 273
Setting the Lisp Comment	Column in Zmacs 227
Default	column position 75
! Dired	command 172
\$ Dired	command 171
, Dired	command 168
. Font Editor drawing	command 300
/ Font Editor	command 330
= Dired	command 169
? Dired	command 167
? Font Editor	command 329
@ Font Editor	command 327
@ text formatting	command 39
@# text formatting	command 39
@* text formatting	command 39
@. text formatting	command 39
@= text formatting	command 39
@> text formatting	command 39
@blankspace text formatting	command 39
@caption text formatting	command 39
@foot text formatting	command 39
@note text formatting	command 39
@tabclear text formatting	command 39
@tabdivide text formatting	command 39
@tabset text formatting	command 39
@\ text formatting	command 39
@^ text formatting	command 39
A Dired	command 172
ABORT Dired	command 167
ABORT Font Editor	command 330
ABORT Zmacs	command 48
Any Extended	Command 7
Append To File (m-X) Zmacs	command 141
Arglist (m-X) Zmacs	command 55
B Font Editor	command 323
C Dired	command 169
C Font Editor	command 298, 330
c-% Zmacs	command 118
c-/ completion	command 15
c-Ø c-m-Y yank	command 15
c-; Zmacs	command 226
c-= Zmacs	command 54
c-? completion	command 15
c-A Zmacs	command 28, 74, 95
c-B Zmacs	command 28, 72
c-C Font Editor	command 329
c-D Dired	command 170
c-D Zmacs	command 30, 49, 90
c-E Zmacs	command 28, 74, 95

c-F Zmacs	command	28, 72
c-G Zmacs	command	48
c-HELP V Zmacs	command	269
c-K Dired	command	170
c-K Zmacs	command	30, 95
c-L Zmacs	command	65
c-m-(Zmacs	command	77
c-m-) Zmacs	command	77
c-m-; Zmacs	command	226
c-m-? V Zmacs	command	269
c-m-@ Zmacs	command	106
c-m-A Zmacs	command	78
c-m-B Zmacs	command	76
c-m-D Zmacs	command	77
c-m-E Zmacs	command	78
c-m-F Zmacs	command	76
c-m-H Zmacs	command	106
c-m-K Zmacs	command	30, 93
c-m-L Zmacs	command	133
c-m-N Zmacs	command	76
c-m-O Zmacs	command	221
c-m-P Zmacs	command	76
c-m-Q Zmacs	command	219
c-m-R Zmacs	command	66
c-m-RUBOUT Zmacs	command	30, 93
c-m-sh-E Zmacs	command	230
c-m-SPACE Zmacs	command	102
c-m-T Zmacs	command	93
c-m-U Zmacs	command	77
c-m-V Zmacs	command	147
c-m-Y yank	command	15
c-m-Z Zmacs	command	230
c-m-[Zmacs	command	78
c-m-\ Zmacs	command	219
c-m-] Zmacs	command	78
c-m-^ Zmacs	command	220
c-N Dired	command	168
c-N Zmacs	command	28, 74, 95, 257
c-O c-Y yank	command	15
c-O Zmacs	command	221
c-P Dired	command	168
c-P Zmacs	command	28, 74, 95
c-R Font Editor	command	329
c-REFRESH Font Editor	command	330
c-sh-A Zmacs	command	55
c-sh-C Zmacs	command	109
c-sh-D Zmacs	command	55
c-sh-E Zmacs	command	230
c-sh-V Zmacs	command	55
c-sh-Y string-matching yank	command	87
c-SPACE Zmacs	command	102
c-T Zmacs	command	90
c-U Zmacs	command	26
c-V Zmacs	command	28, 65
c-W Font Editor	command	329
c-W Zmacs	command	30, 109
c-X) Zmacs	command	257
c-X 1 Zmacs	command	147
c-X 2 Zmacs	command	146
c-X 3 Zmacs	command	146
c-X 4 Zmacs	command	146

c-X 8 Zmacs	command	146
c-X ; Zmacs	command	227
c-X = Zmacs	command	54
c-X A Zmacs	command	141
c-X B Zmacs	command	32, 33, 133
c-X c- ; Zmacs	command	228
c-X c-B Zmacs	command	131, 134
c-X c-D Zmacs	command	149
c-X c-F Zmacs	command	33, 34
c-X c-I Zmacs	command	220
c-X c-L Zmacs	command	214
c-X c-m-L Zmacs	command	133
c-X c-m-SPACE Zmacs	command	102
c-X c-N Zmacs	command	75
c-X c-O Zmacs	command	30, 221
c-X c-P Zmacs	command	107
c-X c-S Zmacs	command	34, 138
c-X c-T Zmacs	command	96
c-X c-U Zmacs	command	214
c-X c-V Zmacs	command	137
c-X c-W Zmacs	command	34, 138
c-X c-X Zmacs	command	103
c-X D Zmacs	command	163
c-X E Zmacs	command	258
c-X F Zmacs	command	32, 254
c-X L Zmacs	command	55
c-X O Zmacs	command	147
c-X Q Zmacs	command	260
c-X RUBOUT Zmacs	command	30, 97
c-X S Zmacs	command	32
c-X T Zmacs	command	109
c-X V Zmacs	command	135
c-X W Zmacs	command	32
c-X Zmacs	command	257
c-X [Zmacs	command	28, 80
c-X] Zmacs	command	28, 80
c-X ^ Zmacs	command	146
c-Y yank	command	15, 87
Cancel last	command	48
Change File Properties (m-X) Zmacs	command	150
Clean Directory (m-X) Zmacs	command	153
CLEAR-INPUT Zmacs	command	96
Compile Region (m-X) Zmacs	command	109
COMPLETE completion	command	15
Copy File (m-X) Zmacs	command	152
Create Directory (m-X) Zmacs	command	148
Create Link (m-X) Zmacs	command	152
Create Spell Dictionary From Namespace	Command	190
D Dired	command	170
D Font Editor	command	323, 329
Deinstall Macro (m-X) Zmacs	command	262
Delete File (m-X) Zmacs	command	153
Describe Variable (m-X) Zmacs	command	269
Dired (m-X) Zmacs	command	163
E Dired	command	169
E Font Editor	command	309, 327
Edit Buffers (m-X) Zmacs	command	134
Edit Cp	Command	238
Edit Definition m- . Zmacs	Command	235
Edit Directory (m-X) Zmacs	command	163
END completion	command	15

END Dired	command	167
End Kbd Macro Zmacs	command	257
Evaluate and Replace Into Buffer (m-X) Zmacs	command	230
Evaluate Buffer (m-X) Zmacs	command	230
Evaluate Changed Definitions (m-X) Zmacs	command	230
Evaluate Changed Definitions of Buffer (m-X) Zmacs	command	230
Evaluate Into Buffer (m-X) Zmacs	command	230
Evaluate Region (m-X) Zmacs	command	230
Example of the m- .	Command	236
Execute Command Into Buffer (m-X) Zmacs	command	136
Execute CP	Command	43
Extended	Command	7
F Font Editor	command	329
(fed) Lisp Listener	command	289
Find File In Fundamental Mode (m-X) Zmacs	command	139
Find File Zmacs	command	33, 34
Finding the right	command	53
Find Unbalanced Parentheses (m-X) Zmacs	command	56
Format Buffer (m-X) Zmacs	command	35, 42
Format File (m-X) Zmacs	command	35, 42
Format Region (m-X) Zmacs	command	35, 42
G Font Editor	command	328
H Font Editor	command	323, 327
Hardcopy Buffer (m-X) Zmacs	command	136
Hardcopy File (m-X) Zmacs	command	151
HELP ? Zmacs	command	14
HELP A Zmacs	command	14, 53
HELP C Zmacs	command	14, 52
HELP completion	command	15
HELP D Zmacs	command	14, 52
HELP Dired	command	167
HELP Font Editor	command	329
HELP L Zmacs	command	14, 53
HELP SPACE Zmacs	command	14
HELP U Zmacs	command	14, 53
HELP V Zmacs	command	14, 54, 269
HELP W Zmacs	command	14
HELP w Zmacs	command	54
Insert Buffer (m-X) Zmacs	command	141
Insert File (m-X) Zmacs	command	141
Install	Command	268
Install Command (m-X) Zmacs	command	268
Install Macro (m-X) Zmacs	command	262
K Dired	command	170
Kill Backward Up List (c-m-X) Zmacs	command	93
L Dired	command	168
LINE Zmacs	command	219
Lisp Mode (m-X) Zmacs	command	224
List Buffers Zmacs	command	131
List Changed Definitions of Buffer (m-X) Zmacs	command	239
List Definitions (m-X) Zmacs	command	239
List Files (m-X) Zmacs	command	148
List Fonts (m-X) Zmacs	command	287
List Variables (m-X) Zmacs	command	269
M Font Editor	command	328
m-% Zmacs	command	118
m-) Zmacs	command	78
m-; Zmacs	command	226
m-< Zmacs	command	28, 81
m-= Zmacs	command	55
m-> Zmacs	command	28, 81

m-@ Zmacs	command	106
m-A Zmacs	command	28
m-B Zmacs	command	28, 72
m-C Zmacs	command	214
m-D Zmacs	command	30, 92
m-E Zmacs	command	73
m-ESCAPE Zmacs	command	230
m-F Zmacs	command	28, 72
m-H Zmacs	command	107
m-K Zmacs	command	30, 97
m-L Zmacs	command	214
m-LINE Zmacs	command	227
m-N Zmacs	command	227
m-O Zmacs	command	220
m-P Zmacs	command	227
m-R Zmacs	command	66
m-RUBOUT Zmacs	command	30, 92
m-S Zmacs	command	219
m-SCROLL Zmacs	command	28, 66
m-sh-D Zmacs	command	54
m-sh-E Zmacs	command	230
m-sh-Y yank	command	88
m-T Zmacs	command	92
m-U Zmacs	command	214
m-V Zmacs	command	28, 66
m-W Zmacs	command	109
m-X	command	210, 212
m-X Edit CP	Command	238
m-Y yank	command	15, 59, 88
m-Z Zmacs	command	231
m-[Zmacs	command	28, 79
m-\ Zmacs	command	30
m-] Zmacs	command	28, 79
m-^ Zmacs	command	30, 220
Name Last Kbd Macro (m-X) Zmacs	command	261
P Dired	command	172
Prepend To File (m-X) Zmacs	command	141
Q Dired	command	167
Q Font Editor	command	329
Query Replace (m-X) Zmacs	command	118
R Dired	command	169
Reap File (m-X) Zmacs	command	153
REFRESH Font Editor	command	330
Rename Buffer (m-X) Zmacs	command	136
Rename File (m-X) Zmacs	command	151
Reparse Attribute List (m-X) Zmacs	command	155
Repeat Last Matching Minibuffer	Command	88
Repeat Last Minibuffer	Command	59, 88
Replace String (m-X) Zmacs	command	118
RETURN completion	command	15
Revert Buffer (m-X) Zmacs	command	138
RUBOUT Dired	command	170
RUBOUT Zmacs	command	30, 49
S Font Editor	command	329
Save File Zmacs	command	34
SCROLL Zmacs	command	28
Selecting a FED Character with the C	Command	298
Set Backspace (m-X) Zmacs	command	159
Set Base (m-X) Zmacs	command	160
Set Fonts (m-X) Zmacs	command	160
Set Lowercase (m-X) Zmacs	command	160

Set Nofill (m-X) Zmacs	command	160
Set Package (m-X) Zmacs	command	156
Set Patch File (m-X) Zmacs	command	161
Set Tab Width (m-X) Zmacs	command	161
Set Variable (m-X) Zmacs	command	271
Set Visited File Name (m-X) Zmacs	command	139
Set Vsp (m-X) Zmacs	command	161
Show Buffer (m-X) Zmacs	command	135
Show Directory (m-X) Zmacs	command	149
Show File (m-X) Zmacs	command	150
Show File Properties (m-X) Zmacs	command	150
Show Keyboard Macro (m-X) Zmacs	command	258
Show Login Directory (m-X) Zmacs	command	149
Source Compare (m-X) Zmacs	command	142
Source Compare Merge (m-X) Zmacs	command	142
SPACE completion	command	15
SPACE Dired	command	168
Trace (m-X) Zmacs	command	56
U Dired	command	170
Update Attribute List (m-X) Zmacs	command	156
V Dired	command	169
V Font Editor	command	321, 329
Variable Apropos Zmacs	command	269
Write File Zmacs	command	34
Zmacs Speller Accept	command	181
Zmacs Speller Accept Once	command	181
Zmacs Speller Prompt	command	181
[Font Editor	command	330
\ Font Editor	command	330
] Font Editor	command	330
Example of Finding Out What a Zmacs	Command Does	51
Finding Out What an Extended	Command Does	52
Finding Out What a Prefix	Command Does	52
Finding Out What a Zmacs	Command Does	51
	Command history	49
Viewing the Editor	Command History	86
Yanking in the	command history	15
Execute	Command Into Buffer	136
Execute	Command Into Buffer (m-X) Zmacs command	136
Extended	command key bindings	54
FED	Command List	327
Zmacs	Command: m-. .	235
Install	Command (m-X) Zmacs command	268
Outside FED	Command Menu	293
	Command menus	292
	Command Names	6
Inserting	Command Output Into the Buffer	136
	Commands	6
Attribute-Manipulating	Commands	155
Basic Text Formatting	Commands	39
Cursor movement	commands	28, 71
Delete	commands	30
Dired	Commands	164, 165
Editor Menu	Commands	57
Evaluation	commands	230
Example of Attribute-Manipulating	Commands	156
Example of Negative Numeric Arguments with Motion	Commands	71
Example of Numeric Arguments with Motion	Commands	71
Extended	commands	6, 52
FED Keyboard-only	Commands	330
FED Menu and Keyboard	Commands	327

Finding Out About Zmacs	Commands 51
General Information-giving Zmacs	Commands 54
Goal Column and the Motion	Commands 74
How to Use Formatting	Commands 38
Init File Form: Fixing White Space for Kill/Yank	Commands 273
Init File Form: Setting Goal Column for Real Line	Commands 273
Introduction to the Character Style	Commands 206
Introduction to the Motion	Commands 71
Introduction to Zmacs	Commands 6
Introduction to Zmacs Extended	Commands 7
Kill	commands 30
List the last sixty	commands 53
Method for Searching for Appropriate Zmacs	Commands 53
More HELP Commands for Finding Out About Zmacs	Commands 53
Motion	Commands 71
Mouse-sensitive Zmacs	commands 67
Names of	commands 7, 52
Negative Numeric Arguments and Motion	Commands 71
Numeric Arguments and the Motion	Commands 71
Online documentation for	commands 52
Other Region-related	Commands 110
Overview of Finding Out About Zmacs	Commands 51
Overview of Zmacs File Manipulation	Commands 148
Prefix	Commands 52
Prefix character	commands 7
Region-Manipulating	Commands 109
Searching for Appropriate	Commands 53
Searching for Appropriate Zmacs	Commands 52
Speller Dictionary	Commands 190
String-matching in yank	commands 85
The Zmacs Edit Callers	Commands 239
The Zmacs Edit Definition	Commands 235
The Zmacs List Definition	Commands 238
Word Abbreviation	Commands 200
Zmacs Buffer	Commands 133
Zmacs File Manipulation	Commands 148
Zmacs Format	Commands 41
Zmacs Sorting	Commands 128
Zmacs Window	Commands 146
Compare/Merge	Commands for Definitions 144
Other Set	Commands for File and Buffer Attributes 159
Set	commands for file and buffer attributes 159
Zmacs	Commands for Finding Out About Flavors 281
Zmacs	Commands for Finding Out About Lisp 280
Zmacs	Commands for Finding Out About the State of Buffers 278
Zmacs	Commands for Finding Out About the State of Zmacs 279
More HELP	Commands for Finding Out About Zmacs Commands 53
Zmacs	Commands for Formatting Text 35
Zmacs	Commands for Interacting with Lisp 282
Speller	Commands for Spelling 184
Executing CP	Commands From Zmacs 43
Character Style	Commands in Zmacs 210
Overview of	Commands to Mark Regions 106
Overview of	Commands to Mark Regions 106
Overview of	Commands to Mark Regions by Buffers 107
Overview of	Commands to Mark Regions by Lisp Expressions 106

- SPACE completion command 15
- How Key Bindings Work: the
 - Standard
 - Font Basic Concepts 285
 - Font Editor Basic Concepts 291
 - Alternative Wide Configuration 291
 - FED Configuration and Drawing Transformation 327
 - Frame configurations 327
 - Deleting the List [Configure] Font Editor menu item 291, 324, 327
 - Creating Two Windows, Specifying Other Containing the Current Lisp Expression 93
 - Viewing and Editing File Contents 146
 - Displaying the Contents in Dired 169
 - List Contents of a Directory 149
 - Retrieving the contents of a directory 148
 - m-X Show Contents of a FED Register 307
 - Writing the Buffer Contents of Spell Dictionary 193
 - Saving the Buffer Contents to a File 138
 - Buffer Contents to the File 138
 - Exclamation point (!) line Contents with c-X c-F 33
- Stretching and continuation indicator 25, 64
 - Using the Contracting a character 327
 - Example 1 of Zmacs Notation Contracting a Drawing Horizontally in FED 316
 - Example 2 of Zmacs Notation Contracting a Drawing Vertically in FED 316
 - Example 3 of Zmacs Notation Contracting Drawings in FED 316
 - Zmacs Manual Notation CONTROL key while drawing characters 299
 - Zmacs Notation Conventions 9
 - Entering Conventions 9
 - Conventions 9
 - Conventions 9
 - Conventions 9
 - Conventions and Examples 9
 - Converse 44
 - Copy File 152
 - Copy File (m-X) Zmacs command 152
- Examples of Copying a File Into Another 151
- Copying a File Into Another 152
- Copying and Renaming Files in Dired 169
- Copying files 169
- Correcting Typos 24
- Spelling correction 180
- Count Characters 61
- Count Chars 61
- Count Lines 61
- Count Lines Page 55
- Count Lines Region 55
- Count Occurrences 62
- Setting Generation Retention Count on Files in Dired 171
- Count Pages 61
- Count Paragraphs 61
- Count Words 61
- Edit Cp Command 238
- Execute CP Command 43
- m-X Edit CP Command 238
- Executing CP Commands From Zmacs 43
- fonts: **cptfont** font 287
- How to Create an Environment 35
- Create Directory 148
- Create Directory (m-X) Zmacs command 148
- Create Link 152
- Create Link (m-X) Zmacs command 152

- Init File Form: Setting Find File Not to
 - Create New Files 273
 - Create Spell Dictionary From Namespace Command 190
 - Creating a Buffer 32, 33
 - Creating a Directory 148
- Example of
 - Creating a Directory 148
 - Creating a File 34
 - Creating a Fundamental Mode Buffer 139
 - Creating and Saving Buffers and Files 32
- Description of
 - Creating and Saving Buffers and Files 32
- Summary of
 - Creating and Saving Buffers and Files 32
- Selecting a FED Character by
 - Creating a New Character 298
 - Creating a New Font in FED 297
 - Creating a New Indented Lisp Comment Line in Zmacs 227
 - Creating an Init File 272
 - Creating a Region 101
 - Creating a Region with Keystrokes 101
 - Creating a Region with the Mouse 101
 - Creating Links to Files 152
 - Creating new characters 298
 - Creating new fonts 297
 - Creating New Zmacs Commands with Keyboard Macros 257
 - Creating Two Windows, Specifying Other Contents 146
 - Creating Two Windows with the Region in Top 146
- Procedure for
 - Creating Zmacs Commands with Keyboard Macros 257
- C-0
 - c-sh-Y 85
- @
 - c text environment 36
- Draw a
 - cubic spline 327
- Comparing/Merging
 - Current/Installed Versions 145
- Comparing/Merging
 - Current/Newest Versions 144
- Comparing/Merging
 - Current/Saved Versions 144
- Current buffer 137
- Deleting the
 - Current Character 90
- Deleting the
 - Current Line 95
- Centering the
 - Current Line in Zmacs 219
- Indenting
 - Current Line in Zmacs 216
- Deleting the
 - Current Lisp Expression 93
- Deleting the List Containing the
 - Current Lisp Expression 93
- Init File Form: Putting Buffers Into
 - Current meaning of mouse clicks 67
 - Current Package 272
 - Current patch 246
 - Current Sentence 97
 - Current Word 92
 - Current Zmacs Buffer 132
- Deleting the
 - Cursor 28
- Deleting the
 - Cursor 28
- Description of Moving the
 - cursor 67
- Introduction to Moving the
 - cursor 299, 300, 330
- Moving the
 - Cursor 64
- Nonmouse
 - cursor 67
- Overview of Moving the
 - Cursor 28
- Relocate
 - Cursor 300
- Summary of Moving the
 - Cursor and Point 18
- The FED Nonmouse
 - Cursor in Zmacs 63
- Editor Window's
 - Cursor Movement 64
- Moving the
 - Cursor movement commands 28, 71
- Summary of
 - cursor to beginning of line 66
- Move
 -

Moving the Cursor with the Mouse 67
 Drawing Lines and Curves in FED 312
 Built-in Customization Using Zmacs Minor Modes 253
 Overview of Customizing the Zmacs Environment 251
 Introduction to Customizing the Zmacs Environment 252
 Introduction to Customizing Zmacs 252
 Customizing Zmacs in Init Files 272
 Customizing Zmacs in Init Files 272

D

D

D

HELP D 52
 SELECT D 44
 D Dired command 170
 D Font Editor command 323, 329
 D Zmacs command 163
 D Zmacs command 14, 52
 c-X Decrypting the Buffer 137
 HELP Default Character Style 212
 Encrypting and Set Default column position 75
 Set Default font 287
 Init File Form: Setting Default major mode 256
 Select Default Major Mode 273
 Move to Default Pathnames in Dired 166
 Base and Syntax Default Previous Buffer 133
 Base and Syntax Default Previous Point 102
 Defaults 157
 One Window Default Settings for Lisp 33, 137, 176, 224
 zwei: Defaults to Numeric Arguments 26
 Beginning of Default syntax 158
 End of Default Variable 270
 Mark **define-keyboard-macro** 259
 Positioning the Window Around a Defining an Interactive Keyboard Macro 260
 Source Compare Installed Definition 78
 Source Compare Merge Installed Definition 78
 Source Compare Merge Newest Definition 106
 Source Compare Merge Saved Definition 66
 Source Compare Newest Definition 145
 Source Compare Saved Definition 145
 The Zmacs Edit Definition 144
 The Zmacs List Definition 144
 Edit Definition 144
 Editing the Definition Commands 235
 Definition Commands 238
 Definition m-. Zmacs Command 235
 definition of a function 12
 Definition of a Zmacs Keyboard Macro 257
 Definition of a Zmacs Variable 269
 Definition of Key Bindings 267
 Definition of Zmacs Minor Modes 253
 Definitions 144
 Compare/Merge Commands for Definitions (m-X) 245
 Add Patch Changed Definitions (m-X) Zmacs command 230
 Evaluate Changed Definitions (m-X) Zmacs command 239
 List Definitions of Buffer (m-X) 245
 Add Patch Changed Definitions of Buffer (m-X) Zmacs command 230
 Evaluate Changed Definitions of Buffer (m-X) Zmacs command 239
 List Changed Definitions of Tag Table 231
 Compile Changed **defmajor** 256
 zwei: Deinstalling a Macro 262

- Example of Installing and Deinstalling a Macro 262
- Deinstall Macro (m-X) Zmacs command 262
- Dired
 - Delete 170
 - Delete Blank Lines 30
 - Delete commands 30
- Protecting Files From Being Deleted in Dired 171
- Delete File 153
- Delete File (m-X) Zmacs command 153
- Delete Forward 30, 90
- Delete Horizontal Space 30
- Delete Indentation 30
- m-X Delete Word From Spell Dictionary 193
- zwei: **delete-words-from-spell-dictionary** function 195
- Deleting and Transposing Characters 90
- Deleting and Transposing Lines 95
- Introduction to Deleting and Transposing Lines 95
- Introduction to Deleting and Transposing Lisp Expressions 93
- Deleting and Transposing Lisp Expressions 93
- Deleting and Transposing Text in Zmacs 83
- Deleting and Transposing Words 92
- Introduction to Deleting and Transposing Words 92
- Deleting a Region 109
- Deleting Backward on the Line 96
- Deleting Blank Line in Zmacs 221
- Deleting Files 153
- Deleting Indentation in Zmacs 220
- Deleting Multiple File Versions in Dired 170
- Deleting Multiple Versions 153
- Deleting Sentences 97
- Introduction to Deleting Sentences 97
- Deleting the Current Character 90
- Deleting the Current Line 95
- Deleting the Current Lisp Expression 93
- Deleting the Current Sentence 97
- Deleting the Current Word 92
- Deleting the Last Character 90
- Deleting the List Containing the Current Lisp Expression 93
- Deleting the Previous Lisp Expression 93
- Deleting the Previous Sentence 97
- Deleting the Previous Word 92
- Deleting Vs. Killing Text 84
- Deleting Vs. Killing Text 84
- Overview of Accidental deletion 49
- Marking Files for Large Deletion in Dired 170
- Description of Zmacs Sentence Deletions 49
- Delimiters 73, 97
- Descenders 302
- Describe Attribute List 168
- Describe Variable 269
- Describe Variable At Point 55
- Describe Variable (m-X) Zmacs command 269
- Describing Zmacs Variables 269
- Description of Creating and Saving Buffers and Files 32
- Description of Erasing Text 30
- Description of Motion by Lisp Expression 76, 93
- Description of Moving the Cursor 28
- Description of Zmacs Sentence Delimiters 73, 97
- Descriptions 159
- Buffer and File Attribute Description text environment 36

- Dired Apply Function 172
- ! Dired command 172
- \$ Dired command 171
- , Dired command 168
- = Dired command 169
- ? Dired command 167
- A Dired command 172
- ABORT Dired command 167
- C Dired command 169
- c-D Dired command 170
- c-K Dired command 170
- c-N Dired command 168
- c-P Dired command 168
- D Dired command 170
- E Dired command 169
- END Dired command 167
- HELP Dired command 167
- K Dired command 170
- L Dired command 168
- P Dired command 172
- Q Dired command 167
- R Dired command 169
- RUBOUT Dired command 170
- SPACE Dired command 168
- U Dired command 170
- V Dired command 169
- Dired Commands 164, 165
- Dired Command Summary 165
- Dired Complement No Reap Flag 171
- Dired Delete 170
- The Dired Display 163
- Updating the Dired Display 164
- Dired Edit File 169
- Dired Exit 167
- Dired Hardcopy File 172
- Dired Help 167
- Dired (m-X) Zmacs command 163
- Dired Menu 167, 168
- Dired Mode in Zmacs 163
- Dired move point 168
- Dired Next Undumped 172
- Dired Reverse Undelete 170
- Dired Srccom 169
- Dired Undelete 170
- Dired View File 169
- Adjusting the FED Display 323
- FED display 289
- Selecting a FED Character From the [Show Font] Display 298
- The Dired Display 163
- Updating the Dired Display 164
- Display argument list 55
- Display Directory 149
- Displaying Characters in the Font in FED 297
- Displaying previous keystrokes 53
- Displaying the Contents of a Directory 149
- Displaying the Next Possibility 126
- Example of Displaying the Next Possibility 127
- Displaying the Next Screen 65
- Displaying the Previous Screen 66
- Mousing on the FED List Fonts and Show Font Displays 332
- Display text environment 36

- Function Documentation 54
- Long Documentation 55
- Show Documentation 54
- Status line documentation 323
- Online documentation for commands 52
- Online Documentation for Dired 167
- Online documentation for prefixes 52
- Mouse Documentation Line 67
- Mouse Documentation Line in Zmacs 67
- Entering Document Examiner 44
- Example of Finding Out What a Zmacs Command Does 51
- Finding Out What an Extended Command Does 52
- Finding Out What a Prefix Command Does 52
- Finding Out What a Zmacs Command Does 51
- Introduction to Tag Tables and Search Domains 122
- Tag Tables and Search Domains in Zmacs 122
- Moving Rest of Line Down in Zmacs 221
- Down Line 74, 95
- Down List 77
- Motion up and Down Nesting Levels 77
- Moving Down Real Line 28, 74, 95
- Down to Lisp Comment on Next Line in Zmacs 227
- Draw a cubic spline 327
- Draw a line 327
- drawing 312, 327
- Reflecting the drawing 309, 327
- Rotating the drawing 309, 327
- Scrolling the drawing 323
- Drawing a character 291
- Using the CONTROL key while drawing characters 299
- Using the META key while drawing characters 299
- Drawing Characters in FED with the Mouse 299
- Drawing characters with the mouse 299
- Font Editor drawing command 300
- Automatic drawing help 309
- Moving the Drawing Horizontally and/or Vertically in FED 323
- Scrolling the Drawing Horizontally and/or Vertically in FED 323
- Contracting a Drawing Horizontally in FED 316
- Stretching a Drawing Horizontally in FED 316
- Drawing in FED 299
- Centering the Drawing in FED 323
- Clearing the Drawing in FED 309
- Moving the Drawing in FED 312
- Positioning the Drawing in FED 323
- Move drawing in the gray plane 303, 328
- Saving a Drawing Into a FED Register 307
- Drawing Lines and Curves in FED 312
- Drawing pane 291, 330
- FED Drawing Pane 291
- Height and width of the drawing pane 324
- Mousing on the FED Drawing Pane 330
- Setting the Box Size in the FED Drawing Pane 324
- Setting the Height and Width of the FED Drawing Pane 324
- Size of boxes in the drawing pane 324, 327
- Using the mouse in the drawing pane 330
- Drawing Pane Menu 292
- Reflecting Drawings in FED 309
- Rotating Drawings in FED 309
- Stretching and Contracting Drawings in FED 316
- FED Configuration and Drawing Transformation 327
- Contracting a Drawing Vertically in FED 316

Stretching a Drawing Vertically in FED 316
 [Draw Line] Font Editor menu item 312, 327
 Draw Mode Menu 292, 299, 331
 Draw Mode Menu 331
 Mousing on the FED draw mode menu 331
 Using the mouse in the draw mode menu item 299
 [Clear Points] Font Editor draw mode menu item 299
 [Flip Points] Font Editor draw mode menu item 299
 [Set Points] Font Editor draw mode menu item 312, 327
 [Draw Spline] Font Editor menu item 312, 327

E

Entering Zmacs with SELECT
 SELECT

c-X

Zmacs

Entering Zmacs with

Bottom

Top

Left and Right

Init File Form:

Init File Form:

The Zmacs

Multiple

m-X

The Zmacs

Dired

Entering Zmacs with **zwei:**

zwei:

Viewing and

Introduction to

Zmacs Major

Locating Source Code to

Entering File System

Font

Font

/ Font

? Font

@ Font

ABORT Font

B Font

E

E 12

E 12, 44

E Dired command 169

E Font Editor command 309, 327

E Zmacs command 258

Echo Area 20

Echo Area 20

Echo Area's Minibuffer 20

Echoing 20

Echoing arguments 26

ed 12

ed function 12

Edge of the FED Character Box 301

Edge of the FED Character Box 301

Edges of the FED Character Box 301

Edit Buffers (m-X) Zmacs command 134

Edit Buffers on c-X c-B 274

Edit Buffers on m-X 275

Edit Callers Commands 239

Edit Callers Intersection 240

Edit Cp Command 238

Edit CP Command 238

Edit Definition Commands 235

Edit Definition m- . Zmacs Command 235

Edit Directory (m-X) Zmacs command 163

Edit File 169

[Edit Font] Font Editor menu item 297, 329

edit-functions 13

edit-functions function 13

Editing a character 291

Editing a File 34

Editing Buffers 134

Editing directories 163

Editing Existing Files 34

Editing File Contents in Dired 169

Editing Lisp Programs in Zmacs 223

Editing Lisp Programs in Zmacs 224

Editing Modes 176

Editing the definition of a function 12

Editing the source code of a function 12

Edit in Zmacs 235

Editor 44

Editor 283

Editor Basic Concepts 291

Editor command 330

Editor command 329

Editor command 327

Editor command 330

Editor command 323

E

C Font	Editor command	298, 330
c-C Font	Editor command	329
c-R Font	Editor command	329
c-REFRESH Font	Editor command	330
c-W Font	Editor command	329
D Font	Editor command	323, 329
E Font	Editor command	309, 327
F Font	Editor command	329
G Font	Editor command	328
H Font	Editor command	323, 327
HELP Font	Editor command	329
M Font	Editor command	328
Q Font	Editor command	329
REFRESH Font	Editor command	330
S Font	Editor command	329
V Font	Editor command	321, 329
[Font	Editor command	330
\ Font	Editor command	330
] Font	Editor command	330
Viewing the	Editor Command History	86
. Font	Editor drawing command	300
[Clear Points] Font	Editor draw mode menu item	299
[Flip Points] Font	Editor draw mode menu item	299
[Set Points] Font	Editor draw mode menu item	299
Overview of the	Editor Menu	57
The	Editor Menu	57
	Editor Menu Commands	57
[Add in Gray] Font	Editor menu item	304, 328
[Center View] Font	Editor menu item	323, 327
[Clear Gray] Font	Editor menu item	303, 328
[Configure] Font	Editor menu item	291, 324, 327
[Draw Line] Font	Editor menu item	312, 327
[Draw Spline] Font	Editor menu item	312, 327
[Edit Font] Font	Editor menu item	297, 329
[Erase All] Font	Editor menu item	303, 309, 327
[EXIT] Font	Editor menu item	329
[Gray Char] Font	Editor menu item	303, 328
[Grid Size] Font	Editor menu item	324, 327
[HELP] Font	Editor menu item	329
[List Fonts] Font	Editor menu item	297, 329
[Move Black] Font	Editor menu item	302, 327
[Move Gray] Font	Editor menu item	303, 328
[Move View] Font	Editor menu item	323, 327
[Read File] Font	Editor menu item	326, 329
[Reflect] Font	Editor menu item	309, 327
[Rename Char] Font	Editor menu item	298, 329
[Rotate] Font	Editor menu item	309, 327
[Save Char] Font	Editor menu item	298, 329
[Set Sample] Font	Editor menu item	321, 329
[Show Font] Font	Editor menu item	291, 297, 298, 329
[Stretch] Font	Editor menu item	327
[Swap Gray] Font	Editor menu item	303, 328
Using the mouse with [List Fonts] Font	Editor menu item	332
Using the mouse with [Show Font] Font	Editor menu item	298, 332
[Write File] Font	Editor menu item	326, 329
Using the mouse with Font	Editor menus	292
Setting	Editor Variables in Init Files	272
Wraparound Lines in the	Editor Window	64
Zmacs	Editor Window	18
The	Editor Window and the Buffer	64
	Editor Window's Buffer	18

- Editor Window's Cursor and Point 18
- Editor Window's Timeout 18
- Edit System Files 238
- Edit Word Abbrevs 201
- Electric P11 Mode 177
- Electric Shift Lock in Lisp Mode 273
- Elements 87
- Elements 87
- Encrypting and Decrypting the Buffer 137
- End 28, 81
- End 107
- END completion command 15
- END Dired command 167
- Ending a Keyboard Macro 258
- End Kbd Macro Zmacs command 257
- End of Buffer 107
- end of buffer 81
- End of Definition 78
- End of Line 28, 74, 95
- Entering and Leaving FED 289
- Entering Converse 44
- Entering Dired 163
- Entering Document Examiner 44
- Entering File System Editor 44
- Entering Flavor Examiner 44
- Entering Inspector 44
- Entering Lisp 44
- Entering Notifications 44
- Entering Peek 44
- Entering Terminal 44
- Entering Zmacs 12, 44
- Entering Zmacs 12
- Entering Zmacs with `ed` 12
- Entering Zmacs with `SELECT E` 12
- Entering Zmacs with the Mouse 12
- Entering Zmacs with `zwl:edit-functions` 13
- Entering Zmail 44
- Enter Minibuffer Responses 59
- enter Zmacs 12
- Enumerate text environment 36
- environment 36
- @b text environment 36
- @c text environment 36
- @g text environment 36
- @i text environment 36
- @p text environment 36
- @r text environment 36
- @t text environment 36
- Boldface text environment 36
- Center text environment 36
- Customizing the Zmacs Environment 251
- Description text environment 36
- Display text environment 36
- Enumerate text environment 36
- Equation text environment 36
- Example text environment 36
- Figure text environment 36
- Flushleft text environment 36
- Flushright text environment 36
- Format text environment 36
- Fullpagefigure text environment 36
- Fullpagetable text environment 36
- Zmacs
 - Init File Form:
 - Retrieving History
 - Using the Mouse on History
 - Goto
 - Mark
- Marking a Region From Here to Moving to
- Introduction to
- More Ways to Using the mouse to

- Heading text environment 36
- How to Create an Environment 36
 - Italics text environment 36
 - Itemize text environment 36
 - Majorheading text environment 36
 - Multiple text environment 36
 - Outputexample text environment 36
- Overview of Customizing the Zmacs Environment 252
 - Quotation text environment 36
 - Subheading text environment 36
 - Table text environment 36
 - Verbatim text environment 36
- Basic Text Formatting Environments 36
 - Equation text environment 36
 - [Erase All] Font Editor menu item 303, 309, 327
 - Erase backward to start of line 96
 - Erasing text 90
- Description of Erasing Text 30
- Introduction to Erasing Text 30
- Summary of Erasing Text 30
- Error recovery 48
- Escaping from prompts 48
- Evaluate And Exit 230
- Evaluate and Replace Into Buffer 120
- Evaluate and Replace Into Buffer (m-X) Zmacs command 230
- Evaluate Buffer (m-X) Zmacs command 230
- Evaluate Changed Definitions (m-X) Zmacs command 230
- Evaluate Changed Definitions of Buffer (m-X) Zmacs command 230
- Evaluate Into Buffer (m-X) Zmacs command 230
- Evaluate Minibuffer 230
- Evaluate Region 230
- Evaluate Region (m-X) Zmacs command 230
- Evaluate Region Verbose 230
- Evaluating and Compiling Lisp Programs in Zmacs 229
- Overview of Evaluating and Compiling Lisp Programs in Zmacs 229
 - Evaluating Forms From FED 330
 - Evaluating Lisp Programs in Zmacs 229
 - Evaluation commands 230
- Entering Document Examiner 44
- Entering Flavor Examiner 44
- Selecting, Listing, and Examining Zmacs Buffers 132
 - Mode Line Example 22
 - Text example 39
- Loop Indentor Example 1 217
 - Example 1 of Making Tables Using Keyboard Macros 264
 - Example 1 of Writing and Saving Keyboard Macros 260
 - Example 1 of Zmacs Notation Conventions 9
- Loop Indentor Example 2 218
 - Example 2 of Making Tables Using Keyboard Macros 265
 - Example 2 of Writing and Saving Keyboard Macros 260
 - Example 2 of Zmacs Notation Conventions 9
 - Example 3 of Zmacs Notation Conventions 9

- Example of a Search String for HELP A 53
- Example of a Source Compare 142
- Example of a Tag Tables Replacement Operation 122
- Example of Attribute-Manipulating Commands 156
- Example of Calling the Last Keyboard Macro 258
- Example of Commands to Mark Regions by Paragraphs 107
- Example of Creating a Directory 148
- Example of Displaying the Next Possibility 127
- Example of Filling Text with Auto Fill Minor Mode 253
- Example of Finding Out What a Zmacs Command Does 51
- Example of Installing and Deinstalling a Macro 262
- Example of Listing Buffers 134
- Example of Listing Variables by Matching a String 270
- Example of Negative Numeric Arguments with Motion Commands 71
- Example of Numeric Arguments 26
- Example of Numeric Arguments with Motion Commands 71
- Example of the m- . Command 236
- Example of Using Tabs to Format Text 41
- Zmacs Notation Conventions and Examples 9
- Examples of Copying a File Into Another 152
- Example text environment 36
- Exchange Lines 96
- Exchange Regions 109
- Exchange Sexps 93
- Exchange Words 92
- Exclamation point (!) line continuation indicator 25, 64
- Execute Command Into Buffer 136
- Execute Command Into Buffer (m-X) Zmacs command 136
- Execute CP Command 43
- Executing CP Commands From Zmacs 43
- Existing Buffer 137
- Existing Files 130
- Existing Files 34
- Exit 167
- Exit 230
- [EXIT] Font Editor menu item 329
- Expanding Lisp Expressions in Zmacs 234
- Expression 93
- Expression 93
- Expression 93
- Expression 76, 93
- Expression 76
- Expression in Zmacs 219
- Expressions 106
- Expressions 93
- Expressions 93
- Expressions 77
- Expressions 93
- Expressions in Zmacs 234
- Expressions in Zmacs 233
- Extended Command 7
- Extended Command 7
- Extended Command Does 52
- Reading a File Into an Buffer Flags for Editing Dired Evaluate And
- Deleting the Current Lisp
- Deleting the List Containing the Current Lisp
- Deleting the Previous Lisp
- Description of Motion by Lisp Motion by Lisp Reindenting
- Commands to Mark Regions by Lisp Deleting and Transposing Lisp
- Introduction to Deleting and Transposing Lisp Motion Among Top-Level Transposing Lisp Expanding Lisp Parenthesizing Lisp
- Any
- Finding Out What an

Extended command key bindings 54
 Extended commands 6, 52
 Introduction to Zmacs Extended Commands 7

F**F****F**

SELECT	F 44
c-X	F Zmacs command 32, 254
	Fast Where Am I 54
Centering the Drawing in	FED 323
Clearing the Drawing in	FED 309
Contracting a Drawing Horizontally in	FED 316
Contracting a Drawing Vertically in	FED 316
Creating a New Font in	FED 297
Displaying Characters in the Font in	FED 297
Drawing in	FED 299
Drawing Lines and Curves in	FED 312
Entering and Leaving	FED 289
Evaluating Forms From	FED 330
Mouse Sensitivities in	FED 330
Moving the Drawing Horizontally and/or Vertically in	FED 323
Moving the Drawing in	FED 312
Positioning the Drawing in	FED 323
Reflecting Drawings in	FED 309
Rotating Drawings in	FED 309
Scrolling the Drawing Horizontally and/or Vertically in	FED 323
Selecting a Character in	FED 297
Selecting a Font in	FED 294
Stretching a Drawing Horizontally in	FED 316
Stretching a Drawing Vertically in	FED 316
Stretching and Contracting Drawings in	FED 316
Transformations on Characters in	FED 309
	FED Character Box 291
Altering the	FED Character Box 302
Bottom Edge of the	FED Character Box 301
Character Height of the	FED Character Box 302
Left and Right Edges of the	FED Character Box 301
Top Edge of the	FED Character Box 301
Viewing and Altering a Character in the	FED Character Box 301
What the Lines Mean in the	FED Character Box 301
Selecting a	FED Character by Creating a New Character 298
Selecting a	FED Character by Renaming Characters 298
Selecting a	FED Character From the Character Select Menu 298
Selecting a	FED Character From the [Show Font] Display 298
	FED Character Select Menu 293
Mousing on the	FED Character Select Pane 331
Selecting a	FED Character with the C Command 298
	FED Command List 327
Outside	FED Command Menu 293
	FED Configuration and Drawing Transformation 327
	FED display 289
Adjusting the	FED Display 323
	FED Drawing Pane 291
Mousing on the	FED Drawing Pane 330
Setting the Box Size in the	FED Drawing Pane 324
Setting the Height and Width of the	FED Drawing Pane 324
Mousing on the	FED Draw Mode Menu 331
Reading	FED Files 325
Reading and Writing	FED Files 325
Writing	FED Files 326

	FED Font Parameters Menu 293
Mousing on the	FED Font Parameters Menu 331
Getting Things Into the	FED Gray Plane 303
Merging Characters with the	FED Gray Plane 304
The	FED Gray Plane 303
	FED Gray Plane Menu Items 328
	FED Keyboard-only Commands 330
	(fed) Lisp Listener command 289
Mousing on the	FED List Fonts and Show Font Displays 332
	FED Menu and Keyboard Commands 327
	FED Menus 292
The	FED Nonmouse Cursor 300
	FED Outside World Interface Menu Items 329
	FED Prompt Pane 292
Retrieving the Contents of a	FED Register 307
Saving a Drawing Into a	FED Register 307
	FED Register Pane 294
Mousing on the	FED Register Pane 331
Retrieving the Black Plane While Manipulating	FED Registers 307
Saving Characters and Pieces of Characters in	FED Registers 307
	FED Sample Pane 291
Mousing on the	FED Sample Pane 331
The	FED Sample String 321
	FED Status Pane 293
	FED, the Subsystem 291
Drawing Characters in	FED with the Mouse 299
	Figure text environment 36
Add region to patch	file 242
Appending a Region to a	File 141
Changing the Properties of a	File 150
Copy	File 152
Creating a	File 34
Creating an Init	File 272
Delete	File 153
Dired Edit	File 169
Dired Hardcopy	File 172
Dired View	File 169
Editing a	File 34
Find	File 32
Format	File 42
Hardcopying a	File 151
Install patch	file 247
m-X Spell	File 185
Naming a	File 138
Prepending a Region to a	File 141
Read Word Abbrev	File 202
Rename	File 151
Renaming a	File 151
Save	File 32, 138
Saving a	File 34
Saving the Buffer Contents to the	File 138
Showing a	File 150
Showing the Properties of a	File 150
Speller dictionary	file 187
View	File 150
Visit	File 137
Write	File 32, 138
Write Word Abbrev	File 203
Writing the Buffer Contents to a	File 138
Other Set Commands for	File and Buffer Attributes 159
Set commands for	file and buffer attributes 159

- Backspace file attribute 159
- Base file attribute 160
- Lowercase file attribute 160
- Nofill file attribute 160
- Patch-File file attribute 161
- Tab-Width file attribute 161
- Vsp file attribute 161
- File Attribute Checking 156
- File Attribute Descriptions 159
- Buffer and Warnings about file attribute lists 156
- File attributes 155
- Viewing File Attributes in Dired 168
- Buffer and File Attributes in Zmacs 155
- File backup flag 172
- File buffers 139
- Viewing and Editing File Contents in Dired 169
- File flags 130
- [Read File] Font Editor menu item 326, 329
- [Write File] Font Editor menu item 326, 329
- Supported file formats 326
- Init File Form: Auto Fill in Text Mode 274
- Init File Form: c-m-L on the SQUARE Key 274
- Init File Form: Edit Buffers on c-X c-B 274
- Init File Form: Edit Buffers on m-X 275
- Init File Form: Electric Shift Lock in Lisp Mode 273
- Init File Form: Fixing White Space for Kill/Yank Commands 273
- Init File Form: m- . on m-(L) 275
- Init File Form: Ordering Buffer Lists 272
- Init File Form: Putting Buffers Into Current Package 272
- Init File Form: Setting Default Major Mode 273
- Init File Form: Setting Find File Not to Create New Files 273
- Init File Form: Setting Goal Column for Real Line Commands 273
- Init File Form: White Space in Lisp Code 274
- Loading a File in Dired 168
- Find File in Fundamental Mode (m-X) Zmacs command 139
- Inserting a File Into a Buffer 141
- Reading a File Into a New Buffer 137
- Reading a File Into an Existing Buffer 137
- Copying a File Into Another 151
- Examples of Copying a File Into Another 152
- Re-reading a File Into the Buffer 138
- Append To File (m-X) Zmacs command 141
- Copy File (m-X) Zmacs command 152
- Delete File (m-X) Zmacs command 153
- Format File (m-X) Zmacs command 35, 42
- Hardcopy File (m-X) Zmacs command 151
- Insert File (m-X) Zmacs command 141
- Prepend To File (m-X) Zmacs command 141
- Reap File (m-X) Zmacs command 153
- Rename File (m-X) Zmacs command 151
- Set Patch File (m-X) Zmacs command 161
- Show File (m-X) Zmacs command 150
- Overview of Zmacs File Manipulation Commands 148
- Zmacs File Manipulation Commands 148
- Set Visited File Name (m-X) Zmacs command 139
- Zmacs Buffer and File Names 130
- Init File Form: Setting Find File Not to Create New Files 273

- File Properties 150
- Show File Properties 150
- View File Properties 150
- Changing File Properties in Dired 168
- Change File Properties (m-X) Zmacs command 150
- Show File Properties (m-X) Zmacs command 150
- Files 32
- files 32
- Association of buffers with Files 130
- Buffer Flags for Existing Files 131
- Buffer Flags for New files 148
- Commands for manipulating files 169
- Copying Files 32
- Creating and Saving Buffers and Files 152
- Creating Links to Files 272
- Customizing Zmacs in Init Files 153
- Deleting Files 32
- Description of Creating and Saving Buffers and Files 34
- Editing Existing Files 238
- Edit System Files 272
- Init Files 273
- Init File Form: Setting Find File Not to Create New Files 272
- Introduction to Customizing Zmacs in Init Files 325
- Reading and Writing FED Files 325
- Reading FED files 326
- Reading font Files 169
- Renaming Files 272
- Setting Editor Variables in Init Files 274
- Setting Key Bindings in Init Files 273
- Setting Mode Hooks in Init Files 32
- Summary of Creating and Saving Buffers and Files 148
- Using the mouse with List Files 326
- Writing FED files 326
- Writing font Files and Buffers in Zmacs 142
- Comparing Files for Deletion in Dired 170
- Marking Files From Being Deleted in Dired 171
- Protecting Files From Being Reaped in Dired 171
- Protecting Files in a Directory 148
- Listing Files in Dired 172
- Applying Arbitrary Functions to Files in Dired 169
- Comparing Recent Versions of Files in Dired 169
- Copying and Renaming Files in Dired 171
- Setting Generation Retention Count on Files in Zmacs 129
- Manipulating Buffers and Files in Zmacs 130
- Overview of Working with Buffers and Files in Zmacs 130
- Working with Buffers and Files (m-X) 136
- Save All Files (m-X) Zmacs command 148
- List Files That Have Not Been Backed up in Dired 172
- Finding Files to Be Hardcopied in Dired 172
- Marking files with buffers 32
- Association of File System Editor 44
- Entering **fs:** ***file-type-mode-alist*** variable 256
- fs:** File Types and Zmacs Major Modes 256
- File versions 130
- file versions 169
- Comparing File Versions in Dired 170
- Deleting Multiple ***file-versions-kept*** variable 153
- zwei:** File with a Buffer 139
- Associating a File Zmacs command 33, 34
- Find File Zmacs command 34
- Save File Zmacs command 34
- Write File Zmacs command 34

- Set Fill Column 254
- Filling a Region 109
- Example of Filling Text with Auto Fill Minor Mode 253
- Init File Form: Auto Fill in Text Mode 274
- Example of Filling Text with Auto Fill Minor Mode 253
- Auto Fill Mode 160, 254
- Fill Paragraph 110
- Set Fill Prefix 110
- Fill Region 110
- Find Character in Style 212
- Find File 32
- Find File In Fundamental Mode (m-X) Zmacs command 139
- Init File Form: Setting Find File Not to Create New Files 273
- Find File Zmacs command 33, 34
- Finding Files That Have Not Been Backed up in Dired 172
- Zmacs Commands for Finding Out About Flavors 281
- Zmacs Commands for Finding Out About Lisp 280
- Zmacs Commands for Finding Out About the State of Buffers 278
- Zmacs Commands for Finding Out About the State of Zmacs 279
- More HELP Commands for Finding Out About Zmacs Commands 51
- Overview of Finding Out About Zmacs Commands 53
- Finding Out About Zmacs Commands 51
- Finding Out About Zmacs Commands with HELP 51
- Finding Out About Zmacs Variables 269
- Finding Out What an Extended Command Does 52
- Finding Out What a Prefix Command Does 52
- Finding Out What a Zmacs Command Does 51
- Example of Finding Out What a Zmacs Command Does 51
- Finding Out What You Have Typed 53
- Finding source code 224
- Finding the right command 53
- Find Unbalanced Parentheses 56
- Find Unbalanced Parentheses (m-X) Zmacs command 56
- Finish Patch (m-X) 247
- Going Back to First Indented Character in Zmacs 219
- Fixed-width* Font Attribute 287
- Fixed-width fonts 287
- Init File Form: Fixing White Space for Kill/Yank Commands 273
- Dired Complement No Reap Flag 171
- File backup flag 172
- Modification flag 130
- + flag in Zmacs 131
- File flags 130
- Buffer Flags for Existing Files 130
- Buffer Flags for New Files 131
- Entering Flavor Examiner 44
- Zmacs Commands for Finding Out About Flavors 281
- [Flip Points] Font Editor draw mode menu item 299
- Flushleft text environment 36
- Flushright text environment 36
- Default font 287
- fonts:cptfont font 287
- Selecting a font 329
- Baseline* Font Attribute 286
- Character Height* Font Attribute 285
- Character Width* Font Attribute 286
- Chars-exist-table* Font Attribute 287
- Fixed-width* Font Attribute 287

- Left Kern* Font Attribute 286
 - Font attributes 285
 - Blinker Width and Blinker Height* Font Attributes 287
 - Font Basic Concepts 285
- Selecting a FED Character From the [Show Mousing on the FED List Fonts and Show] Font] Display 298
- Font Displays 332
- Font Editor 283
- Font Editor Basic Concepts 291
- / Font Editor command 330
- ? Font Editor command 329
- @ Font Editor command 327
- ABORT Font Editor command 330
- B Font Editor command 323
- C Font Editor command 298, 330
- c-C Font Editor command 329
- c-R Font Editor command 329
- c-REFRESH Font Editor command 330
- c-W Font Editor command 329
- D Font Editor command 323, 329
- E Font Editor command 309, 327
- F Font Editor command 329
- G Font Editor command 328
- H Font Editor command 323, 327
- HELP Font Editor command 329
- M Font Editor command 328
- Q Font Editor command 329
- REFRESH Font Editor command 330
- S Font Editor command 329
- V Font Editor command 321, 329
- [Font Editor command 330
- \ Font Editor command 330
-] Font Editor command 330
- . Font Editor drawing command 300
- [Clear Points] Font Editor draw mode menu item 299
- [Flip Points] Font Editor draw mode menu item 299
- [Set Points] Font Editor draw mode menu item 299
- [Add in Gray] Font Editor menu item 304, 328
- [Center View] Font Editor menu item 323, 327
- [Clear Gray] Font Editor menu item 303, 328
- [Configure] Font Editor menu item 291, 324, 327
- [Draw Line] Font Editor menu item 312, 327
- [Draw Spline] Font Editor menu item 312, 327
- [Edit Font] Font Editor menu item 297, 329
- [Erase All] Font Editor menu item 303, 309, 327
- [EXIT] Font Editor menu item 329
- [Gray Char] Font Editor menu item 303, 328
- [Grid Size] Font Editor menu item 324, 327
- [HELP] Font Editor menu item 329
- [List Fonts] Font Editor menu item 297, 329
- [Move Black] Font Editor menu item 302, 327
- [Move Gray] Font Editor menu item 303, 328
- [Move View] Font Editor menu item 323, 327
- [Read File] Font Editor menu item 326, 329
- [Reflect] Font Editor menu item 309, 327
- [Rename Char] Font Editor menu item 298, 329
- [Rotate] Font Editor menu item 309, 327
- [Save Char] Font Editor menu item 298, 329
- [Set Sample] Font Editor menu item 321, 329
- [Show Font] Font Editor menu item 291, 297, 298, 329
- [Stretch] Font Editor menu item 327
- [Swap Gray] Font Editor menu item 303, 328

- Using the mouse with [List Fonts] Font Editor menu item 332
- Using the mouse with [Show Font] Font Editor menu item 298, 332
- [Write File] Font Editor menu item 326, 329
- Using the mouse with Font Editor menus 292
 - Reading font files 326
 - Writing font files 326
 - [Edit] Font Editor menu item 297, 329
 - [Show] Font Editor menu item 291, 297, 298, 329
- Using the mouse with [Show] Font Editor menu item 298, 332
 - Creating a New Font in FED 297
- Displaying Characters in the Font in FED 297
 - Selecting a Font in FED 294
 - Font Parameters menu 293
 - Font Parameters Menu 293
- Mousing on the FED Font Parameters Menu 331
- Using the mouse in the Font Parameters menu 331
 - Attributes of TV Fonts 285
 - Creating new fonts 297
 - Fixed-width fonts 287
 - Introduction to Fonts 285
 - Standard TV Fonts 287
 - Variable-width fonts 286
- Mousing on the FED List fonts:cptfont font 287
 - Fonts and Show Font Displays 332
 - Fonts Attribute 160
- Using the mouse with [List Fonts] Font Editor menu item 297, 329
 - [List Fonts] Font Editor menu item 332
 - List Fonts (m-X) Zmacs command 287
 - Set Fonts (m-X) Zmacs command 160
 - @ foot text formatting command 39
- Minibuffer Response Format 59
 - Format Buffer 42
 - Format Buffer (m-X) Zmacs command 35, 42
 - Format Commands 41
 - Format File 42
 - Format File (m-X) Zmacs command 35, 42
 - Format Region 42
 - Format Region (m-X) Zmacs command 35, 42
- Supported file formats 326
 - Producing Formatted Text 35
- Example of Using Tabs to Format Text 41
 - Format text environment 36
- Inserting Formatting Characters 25
 - @ text formatting command 39
 - @# text formatting command 39
 - @* text formatting command 39
 - @. text formatting command 39
 - @= text formatting command 39
 - @> text formatting command 39
 - @blankspace text formatting command 39
 - @caption text formatting command 39
 - @foot text formatting command 39
 - @note text formatting command 39
 - @tabclear text formatting command 39
 - @tabdivide text formatting command 39
 - @tabset text formatting command 39
 - @\ text formatting command 39
 - ^^ text formatting command 39
- Basic Text Formatting Commands 39
- How to Use Formatting Commands 38
- Basic Text Formatting Environments 36

- Introduction to Text Formatting in Zmacs 35
- Zmacs Commands for Formatting Text 35
- Init File Form: Auto Fill in Text Mode 274
- Init File Form: c-m-L on the SQUARE Key 274
- Init File Form: Edit Buffers on c-X c-B 274
- Init File Form: Edit Buffers on m-X 275
- Init File Form: Electric Shift Lock in Lisp Mode 273
- Init File Form: Fixing White Space for Kill/Yank Commands 273
- Init File Form: m- . on m-(L) 275
- Init File Form: Ordering Buffer Lists 272
- Init File Form: Putting Buffers Into Current Package 272
- Init File Form: Setting Default Major Mode 273
- Init File Form: Setting Find File Not to Create New Files 273
- Init File Form: Setting Goal Column for Real Line Commands 273
- Evaluating Forms From FED 330
- Init File Form: White Space in Lisp Code 274
- Zmacs Fortran Mode 177
- Forward 28
- Delete Forward 30, 90
- Forward Character 72
- Forward List 76
- Forward Page 80
- Forward Paragraph 28, 79
- Forward Sentence 73
- Forward Sexp 76
- Forward up List 77
- Forward Word 28, 72
- Scale fraction 303
- Frame configurations 327
- Protecting Files From Being Deleted in Dired 171
- Protecting Files From Being Reaped in Dired 171
- Evaluating Forms From FED 330
- Marking a Region From Here to Beginning of Buffer 107
- Marking a Region From Here to End of Buffer 107
- Create Spell Dictionary From Namespace Command 190
- Inserting and Removing Lisp Comments From Regions in Zmacs 228
- m-X Delete Word From Spell Dictionary 193
- Selecting a FED Character From the Character Select Menu 298
- Selecting a FED Character From the [Show Font] Display 298
- Executing CP Commands From Zmacs 43
- fs:*file-type-mode-allst*** variable 256
- Fullpagefigure text environment 36
- Fullpagetable text environment 36
- Dired Apply Function 172
- ed** function 12
- Editing the definition of a function 12
- Editing the source code of a function 12
- note-private-patch** function 249
- zwei:add-words-to-spell-dictionary** function 194
- zwei:delete-words-from-spell-dictionary** function 195
- zwei:edit-function** function 13
- zwei:read-spell-dictionary** function 193
- zwei:read-standard-spell-dictionaries** function 194
- Function Documentation 54
- Speller Dictionary Functions 193
- Applying Arbitrary Functions to Files in Dired 172
- Zmacs Fundamental Mode 176
- Creating a Fundamental Mode Buffer 139
- Find File In Fundamental Mode (m-X) Zmacs command 139

G

G

G

- G Font Editor command 328
- General Information-giving Zmacs Commands 54
- Setting Generation Retention Count on Files in Dired 171
- Getting Help in Zmacs 47
- Getting Information About Buffers and Regions 61
- Getting Out of Dired 166
- Getting Out of Keystroke Prefixes 48
- Getting Out of Minibuffer Prompts 48
- Getting Out of Prefixes and Prompts 48
- Getting Out of Trouble 48
- Overview of Getting Out of Trouble 48
- Getting Started in Zmacs 11
- Getting Text Back 49
- Getting Things Into Gray with [Gray Char] 303
- Getting Things Into Gray with [Swap Gray] 303
- Getting Things Into the FED Gray Plane 303
- Making Global Replacements in Zmacs 118
- Querying While Making Global Replacements in Zmacs 118
- Querying While Making Multiple Global Replacements in Zmacs 119
- zibase global variable 160
- zwel:*set-attribute-update-list* global variable 159
- c-X plus-SIGN Add Global Word Abbrev 200
- Set Goal Column 75
- Init File Form: Setting Goal Column and the Motion Commands 74
- Goal Column for Real Line Commands 273
- Going Back to First Indented Character in Zmacs 219
- Goto Beginning 28, 81
- Goto End 28, 81
- Mouse as a graphic input device 291
- Getting Things Into Gray with [Swap Gray] 303
- Getting Things Into Gray with [Gray Char] 303
- [Add in [Gray Char] Font Editor menu item 303, 328
- [Clear Gray] Font Editor menu item 304, 328
- [Move Gray] Font Editor menu item 303, 328
- [Swap Gray] Font Editor menu item 303, 328
- Clear Gray plane 293, 303
- Getting Things Into the FED gray plane 303, 328
- Merging Characters with the FED Gray Plane 303
- Move drawing in the Gray Plane 304
- The FED gray plane 303, 328
- FED Gray Plane 303
- Getting Things Into Gray Plane Menu 293
- Getting Things Into Gray Plane Menu Items 328
- Getting Things Into Gray with [Gray Char] 303
- Getting Things Into Gray with [Swap Gray] 303
- [Grid Size] Font Editor menu item 324, 327
- Grow Window 146
- @ g text environment 36

H

H

H

- H Font Editor command 323, 327
- Marking Files to Be Hardcopied in Dired 172
- Dired Hardcopy Buffer (m-X) Zmacs command 136
- Hardcopy File 172
- Hardcopy File (m-X) Zmacs command 151
- Hardcopying a File 151
- Hardcopying a Region 109

Finding Files That Have Not Been Backed up in Dired 172
 Finding Out What You Have Typed 53
 Text heading 39
 Heading text environment 36
 height 287
 Blinker height 285, 291, 302
 Character Line height 285
 Character Height Font Attribute 285
 Blinker Width and Blinker Height Font Attributes 287
 Setting the Height and width of the drawing pane 324
 Character Height and Width of the FED Drawing Pane 324
 Height of the FED Character Box 302
 HELP 51
 Automatic drawing help 309
 Dired Help 167
 Finding Out About Zmacs Commands with HELP 51
 Introduction to HELP 14
 Introduction to Zmacs Help 14
 Minibuffer Response Help 59
 Zmacs Help 14
 Example of a Search String for HELP ? Zmacs command 14
 HELP A 53
 HELP A Zmacs command 14, 53
 HELP C Zmacs command 14, 52
 More HELP Commands for Finding Out About Zmacs
 Commands 53
 HELP completion command 15
 HELP D Zmacs command 14, 52
 HELP Dired command 167
 HELP Font Editor command 329
 HELP key 14, 51
 HELP L Zmacs command 14, 53
 HELP or c-? 15
 HELP SPACE Zmacs command 14
 HELP U Zmacs command 14, 53
 HELP V Zmacs command 14, 54, 269
 HELP W Zmacs command 54
 HELP W Zmacs command 14
 HELP a 52
 HELP C 51
 Zmacs Help Command Summary 277
 HELP D 52
 [HELP] Font Editor menu item 329
 Getting Help in Zmacs 47
 HELP L 53
 HELP U 53
 HELP V 54
 HELP W 54
 Marking a Region From Here to Beginning of Buffer 107
 Marking a Region From Here to End of Buffer 107
 What Histories Save 84
 Buffer History 132
 Command history 49
 Kill history 49, 85, 89
 Viewing the Editor Command History 86
 Viewing the Kill History 85
 Yanking in the command history 15
 Yanking in the kill history 15
 Zmacs Buffer History 132
 Retrieving History Elements 87

- Init File Form: Putting Buffers Into Current Package 272
- Init File Form: Setting Default Major Mode 273
- Init File Form: Setting Find File Not to Create New Files 273
- Init File Form: Setting Goal Column for Real Line Commands 273
- Init File Form: White Space in Lisp Code 274
- Init Files 272
- Init Files 272
- Init Files 272
- Init Files 272
- Init Files 274
- Init Files 273
- Initial patch state 246
- In-progress patch 246
- In-progress patch state 246
- input device 291
- Insert Buffer (m-X) Zmacs command 141
- Insert File (m-X) Zmacs command 141
- Inserting a Buffer Into Another Buffer 141
- Inserting a File Into a Buffer 141
- Inserting and Removing Lisp Comments From Regions in Zmacs 228
- Inserting Blank Line in Zmacs 221
- Inserting Characters 24
- Inserting Command Output Into the Buffer 136
- Inserting Formatting Characters 25
- Inserting output into the buffer 136
- Inserting Regions in Registers 104
- Inserting Text 24
- Inserting Text 24
- Inserting Text in Zmacs 141
- Insert Matching parentheses 233
- Insert text from register into buffer 105
- Insert Word Abbrevs 201
- Inspector 44
- Install Command 268
- Install Command (m-X) Zmacs command 268
- Installed Definition 145
- Installed Definition 145
- Installing a Macro on a Key 261
- Installing a Mouse Macro 262
- Installing and Deinstalling a Macro 262
- Install Macro (m-X) Zmacs command 262
- Install patch file 247
- Interacting with Lisp 282
- Interactive Keyboard Macro 260
- Interface Menu Items 329
- Intersection 240
- Intersection 241
- Introduction to Completion 15
- Introduction to Customizing Zmacs 252
- Introduction to Customizing Zmacs in Init Files 272
- Introduction to Deleting and Transposing Lines 95
- Introduction to Deleting and Transposing Lisp Expressions 93
- Introduction to Deleting and Transposing Words 92
- Introduction to Deleting Sentences 97
- Introduction to Editing Lisp Programs in Zmacs 224
- Introduction to Entering Zmacs 12
- Customizing Zmacs in
- Introduction to Customizing Zmacs in
- Setting Editor Variables in
- Setting Key Bindings in
- Setting Mode Hooks in
- Mouse as a graphic
- Saving and
- Introduction to
- Appending, Prepending, and
- Entering
- Source Compare
- Source Compare Merge
- Example of
- Zmacs Commands for
- Defining an
- FED Outside World
- Multiple Edit Callers
- Multiple List Callers

Introduction to Erasing Text 30
 Introduction to Fonts 285
 Introduction to HELP 14
 Introduction to Inserting Text 24
 Introduction to Locating Source Code in Zmacs 235
 Introduction to Motion by Page 80
 Introduction to Motion by Paragraph 79
 Introduction to Moving the Cursor 28
 Introduction to Redisplaying the Window 65
 Introduction to Regions 100
 Introduction to Speller Dictionaries 187
 Introduction to Tag Tables and Search Domains 122
 Introduction to Text Formatting in Zmacs 35
 Introduction to the Character Style Commands 206
 Introduction to the Motion Commands 71
 Introduction to the Organization of the Screen 18
 Introduction to the Zmacs Manual 3
 Introduction to Using the Mouse 67
 Introduction to Yanking 15
 Introduction to Zmacs 6
 Introduction to Zmacs Commands 6
 Introduction to Zmacs Command Tables 7
 Introduction to Zmacs Extended Commands 7
 Introduction to Zmacs Help 14
 Introduction to Zmacs Keystrokes 7
 Invoking Zmacs 12
 What is a Zmacs Region? 100
 Italics text environment 36
 Itemize text environment 36
 Items 328
 Items 329
 FED Gray Plane Menu
 FED Outside World Interface Menu

K

K

K

K Dired command 170
 Start Kbd Macro 257
 Name Last Kbd Macro (m-X) Zmacs command 261
 Kbd Macro Query 260
 End Kbd Macro Zmacs command 257
 Left kern 286
 Left Kern Font Attribute 286
 HELP key 14, 51
 Init File Form: c-m-L on the SQUARE Key 274
 Installing a Macro on a Key 261
 Leaving Zmacs with the SELECT Key 44
 RUBOUT key 24
 SELECT key 44
 Setting the Key 267
 Assign key bindings 267
 Definition of Key Bindings 267
 Extended command key bindings 54
 Zmacs Key Bindings 267
 Setting Key Bindings in Init Files 274
 How Key Bindings Work: the Comtab 267
 FED Menu and Keyboard Commands 327
 Calling the Last Keyboard Macro 258
 Defining an Interactive Keyboard Macro 260
 Definition of a Zmacs Keyboard Macro 257
 Ending a Keyboard Macro 258
 Example of Calling the Last Keyboard Macro 258
 Naming a Keyboard Macro 261

Showing a Keyboard Macro 258
 Starting a Keyboard Macro 258
 Show Keyboard Macro (m-X) Zmacs command 258
 Creating New Zmacs Commands with Keyboard Macros 257
 Example 1 of Making Tables Using Keyboard Macros 264
 Example 1 of Writing and Saving Keyboard Macros 260
 Example 2 of Making Tables Using Keyboard Macros 265
 Example 2 of Writing and Saving Keyboard Macros 260
 Making Tables Using Keyboard Macros 263
 Procedure for Creating Zmacs Commands with Keyboard Macros 257
 Sort Via Keyboard Macros 261
 Writing and Saving Keyboard Macros 259
 Using Keyboard Macros to Sort 261
 How Zmacs Keyboard Macros Work 257
 FED Keyboard-only Commands 330
 RETURN key in the minibuffer 59
 Shift keys 7
 Getting Out of Keystroke Prefixes 48
 Keystrokes 6, 7
 Creating a Region with Keystrokes 101
 Displaying previous keystrokes 53
 Introduction to Zmacs Keystrokes 7
 List the last sixty keystrokes 53
 Using the CONTROL key while drawing characters 299
 Using the META key while drawing characters 299
 Append Next Kill 89
 Query Replace Last Kill 120
 Init File Form: Fixing White Space for Kill/Yank Commands 273
 Kill All Word Abbrevs 201
 Kill Backward Up List (c-m-X) Zmacs command 93
 Kill commands 30
 Kill history 49, 85, 89
 Viewing the Kill History 85
 Yanking in the kill history 15
 Killing a Lisp Comment in Zmacs 226
 Deleting Vs. Killing Text 84
 Overview of Deleting Vs. Killing Text 84
 Kill Line 30, 95
 Kill Merging 89
 Kill Region 30
 Kill Sentence 30, 97
 Backward Kill Sentence 30
 Kill Sexp 30, 93
 Backward Kill Sexp 30, 93
 m-X Kill Spell Dictionary 192
 Kill Word 30, 92
 Backward Kill Word 30, 92

L

L

L

HELP L 53
 Init File Form: m-. on m- (L) 275
 SELECT L 44
 c-X L Dired command 168
 HELP L Zmacs command 55
 L Zmacs command 14, 53
 Large Deletions 49
 Last Character 90
 Deleting the last command 48
 Cancel Name Last Kbd Macro (m-X) Zmacs command 261
 Calling the Last Keyboard Macro 258

- Example of Calling the Last Keyboard Macro 258
- Query Replace Last Kill 120
- Repeat Last Matching Minibuffer Command 88
- Repeat Last Minibuffer Command 59, 88
- List the last sixty commands 53
- List the last sixty keystrokes 53
- c-X U Unexpand Last Word 203
- Entering and Leaving FED 289
- Overview of Leaving Zmacs 44
- Leaving Zmacs 44
- Leaving Zmacs Via the System Menu 44
- Leaving Zmacs with c-Z 45
- Leaving Zmacs with the SELECT Key 44
- Left and Right Edges of the FED Character Box 301
- Left kern 286
- Left Kern* Font Attribute 286
- Query Replace LET Binding 121
- Abort At Top Level 48
- Motion Along One Nesting Level 76
- Motion up and Down Nesting Levels 77
- Beginning of Line 28, 74, 95
- Breaking a line 24
- Deleting Backward on the Line 96
- Deleting the Current Line 95
- Down Line 74, 95
- Down Real Line 28, 74, 95
- Draw a line 327
- End of Line 28, 74, 95
- Erase backward to start of line 96
- Kill Line 30, 95
- Motion by Line 74
- Mouse Documentation Line 67
- Move cursor to beginning of line 66
- Moving to a Specified Line 66
- Starting a New Line 24
- Up Line 74, 95
- Up Real Line 28, 74, 95
- Zmacs Mode Line 20
- LINE Zmacs command 219
- Init File Form: Setting Goal Column for Real Line Commands 273
- Exclamation point (!) line continuation indicator 25, 64
- Status line documentation 323
- Moving Rest of Line Down in Zmacs 221
- Mode Line Example 22
- [Draw Line] Font Editor menu item 312, 327
- Line height 285
- Centering the Current Line in Zmacs 219
- Creating a New Indented Lisp Comment Line in Zmacs 227
- Deleting Blank Line in Zmacs 221
- Indenting Current Line in Zmacs 216
- Indenting New Line in Zmacs 219
- Inserting Blank Line in Zmacs 221
- Mouse Documentation Line in Zmacs 67
- Moving Down to Lisp Comment on Next Line in Zmacs 227
- Moving up to Lisp Comment on Previous Line in Zmacs 227
- Count Lines 61
- Delete Blank Lines 30
- Deleting and Transposing Lines 95
- Exchange Lines 96
- Introduction to Deleting and Transposing Lines 95
- Merging lines 24

- Wraparound Lines 64
- Wrapping Lines 25
 - Drawing Lines and Curves in FED 312
 - Mode Line's *Buffer* 21
 - Mode Line's *Buffer-status* 22
- Wraparound Lines in the Editor Window 64
 - Mode Line's *Major-mode* 21
- What the Lines Mean in the FED Character Box 301
 - Mode Line's *Minor-mode* 21
- Transposing Lines of Text 96
 - Count Lines Page 55
 - Mode Line's *Position-flag* 22
 - Count Lines Region 55
 - Mode Line's *Version* 22
 - New Line with This Indentation in Zmacs 220
 - Create Link 152
 - Create Link (m-X) Zmacs command 152
 - Creating Links to Files 152
- Base and Syntax Default Settings for Lisp 33, 137, 176, 224
 - Entering Lisp 44
- Setting the Syntax for Symbolics Common Lisp 158
- Zmacs Commands for Finding Out About Lisp 280
 - Zmacs Commands for Interacting with Lisp 282
 - Init File Form: White Space in Lisp Code 274
 - Commenting Lisp Code in Zmacs 226
 - Overview of Commenting Lisp Code in Zmacs 226
 - Setting the Lisp Comment Column in Zmacs 227
 - Indenting for Lisp Comment in Zmacs 226
 - Killing a Lisp Comment in Zmacs 226
 - Creating a New Indented Lisp Comment Line in Zmacs 227
 - Moving Down to Lisp Comment on Next Line in Zmacs 227
 - Moving up to Lisp Comment on Previous Line in Zmacs 227
 - Lisp comments 228
 - Inserting and Removing Lisp Comments From Regions in Zmacs 228
 - Lisp Compiler Warnings 232
 - Deleting the Current Lisp Expression 93
 - Deleting the List Containing the Current Lisp Expression 93
 - Deleting the Previous Lisp Expression 93
 - Description of Motion by Lisp Expression 76, 93
 - Motion by Lisp Expression 76
 - Commands to Mark Regions by Lisp Expressions 106
 - Deleting and Transposing Lisp Expressions 93
 - Introduction to Deleting and Transposing Lisp Expressions 93
 - Transposing Lisp Expressions 93
 - Expanding Lisp Expressions in Zmacs 234
 - Parenthesizing Lisp Expressions in Zmacs 233
 - (fed) Lisp Listener command 289
 - Init File Form: Electric Shift Lock in Lisp Mode 273
 - Zmacs Lisp Mode 176
 - Lisp Mode (m-X) Zmacs command 224
 - Compiling Lisp Programs in Zmacs 230
 - Editing Lisp Programs in Zmacs 223
 - Evaluating Lisp Programs in Zmacs 229
 - Evaluating and Compiling Lisp Programs in Zmacs 229
 - Introduction to Editing Lisp Programs in Zmacs 224
 - Overview of Evaluating and Compiling Lisp Programs in Zmacs 229
 - Attribute list 155
 - Backward List 76
 - Backward up List 77
 - Describe Attribute List 168
 - Display argument list 55

- Down List 77
- FED Command List 327
- Forward List 76
- Forward up List 77
- History list 15
- Speller dictionary list 187
 - List Buffers 134
 - List Buffers Zmacs command 131
- Kill Backward Up List (c-m-X) Zmacs command 93
 - Multiple List Callers Intersection 241
 - List Changed Definitions of Buffer (m-X) Zmacs command 239
- Deleting the List Containing the Current Lisp Expression 93
 - The Zmacs List contents of a directory 148
 - List Definition Commands 238
 - List Definitions (m-X) Zmacs command 239
- (fed) Lisp Listener command 289
- Using the mouse with List Files 148
 - List Files (m-X) Zmacs command 148
- Mousing on the FED List Fonts and Show Font Displays 332
 - [List Fonts] Font Editor menu item 297, 329
- Using the mouse with [List Fonts] Font Editor menu item 332
 - List Fonts (m-X) Zmacs command 287
 - Selecting, Listing, and Examining Zmacs Buffers 132
 - Example of Listing Buffers 133
 - Listing Buffers 134
 - Listing Files in a Directory 148
 - Specifying and Listing Tag Tables 122
 - Example of Listing Variables by Matching a String 269
 - Listing Variables by Matching a String 270
 - Listing Zmacs Variables 269
- Reparse Attribute List (m-X) Zmacs command 155
- Update Attribute List (m-X) Zmacs command 156
 - List of dictionaries 193
 - Speller list of dictionaries 187
- Update Attribute List Query 159
- Attribute lists 156
- Init File Form: Ordering Buffer Lists 272
- Warnings about file attribute lists 156
 - List Some Word Abbrevs 202
 - List the last sixty commands 53
 - List the last sixty keystrokes 53
 - List Variables 269
 - List Variables (m-X) Zmacs command 269
 - List Word Abbrevs 202
 - L:Move point mouse click 67
 - Loading a File in Dired 168
- Overview of Locating and Replacing Strings Automatically 118
 - Locating and Replacing Strings Automatically in Zmacs 118
- Introduction to Locating Source Code in Zmacs 235
 - Locating Source Code to Edit in Zmacs 235
- Saving and Moving to Locations in Registers 104
- Init File Form: Electric Shift Lock in Lisp Mode 273
 - Show Login Directory 149
 - Show Login Directory (m-X) Zmacs command 149
- Long Documentation 55
- How to Use the zl: loop Indentor 217
 - The zl: loop Indentor 216
 - TAB in zl: loop macro 216
 - Indentation in zl: loop Macros 216

Loop Indenter Example 1 217
 Loop Indenter Example 2 218
 Lowercase Attribute 160
 Lowercase file attribute 160
 Set Lowercase (m-X) Zmacs command 160

M**M****M**

SELECT M 44
 M Font Editor command 328
 m-\$ 180
 m-\$ (Spell This Word) 184
 m-% Zmacs command 118
 m-) Zmacs command 78
 Zmacs Command:
 Example of the m- . 235
 Init File Form: m- . Command 236
 Edit Definition m- . on m-(L) 275
 m- . Zmacs Command 235
 m-; Zmacs command 226
 m-< Zmacs command 28, 81
 m-= Zmacs command 55
 m-> Zmacs command 28, 81
 m-@ Zmacs command 106
 m-A Zmacs command 28
 m-B Zmacs command 28, 72
 m-C Zmacs command 214
 m-D Zmacs command 30, 92
 m-E Zmacs command 73
 m-ESCAPE Zmacs command 230
 m-F Zmacs command 28, 72
 m-H Zmacs command 107
 m-J Change Style Word 211
 m-K Zmacs command 30, 97
 Init File Form: m- . on m-(L) 275
 m-L Zmacs command 214
 m-LINE Zmacs command 227
 m-N Zmacs command 227
 m-O Zmacs command 220
 m-P Zmacs command 227
 m-R Zmacs command 66
 m-RUBOUT Zmacs command 30, 92
 m-S Zmacs command 219
 m-SCROLL Zmacs command 28, 66
 m-sh-D Zmacs command 54
 m-sh-E Zmacs command 230
 m-sh-Y yank command 88
 m-T Zmacs command 92
 m-U Zmacs command 214
 m-V Zmacs command 28, 66
 m-W Zmacs command 109
 m-X 7
 Abort Patch (m-X) 248
 Add Patch (m-X) 244
 Add Patch Changed Definitions (m-X) 245
 Add Patch Changed Definitions of Buffer (m-X) 245
 Apropos (m-X) 52
 Arglist (m-X) 55
 Finish Patch (m-X) 247
 Init File Form: Edit Buffers on m-X 275
 Recompile Patch (m-X) 248
 Reload Patch (m-X) 248

Resume Patch (m-X)	248
Save All Files (m-X)	136
Select Patch (m-X)	246
Set Syntax (m-X)	158
Show Patches (m-X)	246
Start Patch (m-X)	243
Start Private Patch (m-X)	244
	m-X Add Word to Spell Dictionary	192
	m-X Apropos	53
	m-X command	210, 212
	m-X Compile Spell Dictionary	190
	m-X Delete Word From Spell Dictionary	193
	m-X Edit CP Command	238
	m-X Kill Spell Dictionary	192
	m-X Read Spell Dictionary	190
	m-X Save All Spell Dictionaries	192
	m-X Save Spell Dictionary	191
	m-X Show Contents of Spell Dictionary	193
	m-X Show Spell Dictionaries	191
	m-X Spell Buffer	180, 185
	m-X Spell File	185
	m-X Spell Region	184
	m-X Spell Word	184
	m-X Tags Spell	185
Append To File (m-X) Zmacs command	141
Arglist (m-X) Zmacs command	55
Change File Properties (m-X) Zmacs command	150
Clean Directory (m-X) Zmacs command	153
Compile Region (m-X) Zmacs command	109
Copy File (m-X) Zmacs command	152
Create Directory (m-X) Zmacs command	148
Create Link (m-X) Zmacs command	152
Deinstall Macro (m-X) Zmacs command	262
Delete File (m-X) Zmacs command	153
Describe Variable (m-X) Zmacs command	269
Dired (m-X) Zmacs command	163
Edit Buffers (m-X) Zmacs command	134
Edit Directory (m-X) Zmacs command	163
Evaluate and Replace Into Buffer (m-X) Zmacs command	230
Evaluate Buffer (m-X) Zmacs command	230
Evaluate Changed Definitions (m-X) Zmacs command	230
Evaluate Changed Definitions of Buffer (m-X) Zmacs command	230
Evaluate Into Buffer (m-X) Zmacs command	230
Evaluate Region (m-X) Zmacs command	230
Execute Command Into Buffer (m-X) Zmacs command	136
Find File In Fundamental Mode (m-X) Zmacs command	139
Find Unbalanced Parentheses (m-X) Zmacs command	56
Format Buffer (m-X) Zmacs command	35, 42
Format File (m-X) Zmacs command	35, 42
Format Region (m-X) Zmacs command	35, 42
Hardcopy Buffer (m-X) Zmacs command	136
Hardcopy File (m-X) Zmacs command	151
Insert Buffer (m-X) Zmacs command	141
Insert File (m-X) Zmacs command	141
Install Command (m-X) Zmacs command	268
Install Macro (m-X) Zmacs command	262
Lisp Mode (m-X) Zmacs command	224
List Changed Definitions of Buffer (m-X) Zmacs command	239
List Definitions (m-X) Zmacs command	239
List Files (m-X) Zmacs command	148
List Fonts (m-X) Zmacs command	287

- List Variables (m-X) Zmacs command 269
- Name Last Kbd Macro (m-X) Zmacs command 261
- Prepend To File (m-X) Zmacs command 141
- Query Replace (m-X) Zmacs command 118
- Reap File (m-X) Zmacs command 153
- Rename Buffer (m-X) Zmacs command 136
- Rename File (m-X) Zmacs command 151
- Reparse Attribute List (m-X) Zmacs command 155
- Replace String (m-X) Zmacs command 118
- Revert Buffer (m-X) Zmacs command 138
- Set Backspace (m-X) Zmacs command 159
- Set Base (m-X) Zmacs command 160
- Set Fonts (m-X) Zmacs command 160
- Set Lowercase (m-X) Zmacs command 160
- Set Nofill (m-X) Zmacs command 160
- Set Package (m-X) Zmacs command 156
- Set Patch File (m-X) Zmacs command 161
- Set Tab Width (m-X) Zmacs command 161
- Set Variable (m-X) Zmacs command 271
- Set Visited File Name (m-X) Zmacs command 139
- Set Vsp (m-X) Zmacs command 161
- Show Buffer (m-X) Zmacs command 135
- Show Directory (m-X) Zmacs command 149
- Show File (m-X) Zmacs command 150
- Show File Properties (m-X) Zmacs command 150
- Show Keyboard Macro (m-X) Zmacs command 258
- Show Login Directory (m-X) Zmacs command 149
- Source Compare (m-X) Zmacs command 142
- Source Compare Merge (m-X) Zmacs command 142
- Trace (m-X) Zmacs command 56
- Update Attribute List (m-X) Zmacs command 156
- m-Y yank command 15, 59, 88
- m-Z Zmacs command 231
- m-[Zmacs command 28, 79
- m-\ Zmacs command 30
- m-] Zmacs command 28, 79
- m-^ Zmacs command 30, 220
- Calling the Last Keyboard Macro 258
- Defining an Interactive Keyboard Macro 260
- Definition of a Zmacs Keyboard Macro 257
- Deinstalling a Macro 262
- Ending a Keyboard Macro 258
- Example of Calling the Last Keyboard Macro 258
- Example of Installing and Deinstalling a Macro 262
- Installing a Mouse Macro 262
- Naming a Keyboard Macro 261
- Showing a Keyboard Macro 258
- Starting a Keyboard Macro 258
- Start Kbd Macro 257
- TAB in z!:loop macro 216
- Deinstall Macro (m-X) Zmacs command 262
- Install Macro (m-X) Zmacs command 262
- Name Last Kbd Macro (m-X) Zmacs command 261
- Show Keyboard Macro (m-X) Zmacs command 258
- Installing a Macro on a Key 261
- Kbd Macro Query 260
- Creating New Zmacs Commands with Keyboard Macros 257
- Example 1 of Making Tables Using Keyboard Macros 264
- Example 1 of Writing and Saving Keyboard Macros 260
- Example 2 of Making Tables Using Keyboard Macros 265
- Example 2 of Writing and Saving Keyboard Macros 260

Indentation in zl:loop	Macros 216
Making Tables Using Keyboard	Macros 263
Procedure for Creating Zmacs Commands with Keyboard	Macros 257
Sort Via Keyboard	Macros 261
Writing and Saving Keyboard	Macros 259
Using Keyboard	Macros to Sort 261
How Zmacs Keyboard	Macros Work 257
End Kbd	Macro Zmacs command 257
Zmacs	Macsyma Mode 177
Send	mail about patch 247
Zmacs	Major Editing Modes 176
Default	Majorheading text environment 36
Init File Form: Setting Default	major mode 256
Mode Line's	Major Mode 273
Overview of Setting the	<i>Major-mode</i> 21
Setting the Zmacs	Major Mode 176
File Types and Zmacs	Major Mode 175
User-Defined Zmacs	Major Modes 256
Zmacs	Major Modes 256
zwel:	*major-mode-translations* variable 256
zwel:	make-macro-command 259
	Make region 67
	Make Word Abbrev 202
	Making Global Replacements in Zmacs 118
Querying While	Making Global Replacements in Zmacs 118
Querying While	Making Multiple Global Replacements in Zmacs 119
	Making Patches 242
	Making Tables Using Keyboard Macros 263
Example 1 of	Making Tables Using Keyboard Macros 264
Example 2 of	Making Tables Using Keyboard Macros 265
Speller Dictionary	Management 187
	Manipulating Buffers and Files in Zmacs 129
Retrieving the Black Plane While	Manipulating FED Registers 307
Commands for	manipulating files 148
Overview of Zmacs File	Manipulation Commands 148
Zmacs File	Manipulation Commands 148
Introduction to the Zmacs	Manual 3
Organization of the Zmacs	Manual 4
Overview of the Zmacs	Manual 4
Scope of the Zmacs	Manual 4
Zmacs	Manual 1
Zmacs	Manual Notation Conventions 9
How	Many 62
Right	margin 254
Region Right	Margin Mode Variable 270
Set Pop	Mark 102
Setting/Popping the	Mark 102
Showing the	Mark 103
Swap Point And	Mark 103
	Mark and the Region 100
	Mark Beginning 107
	Mark Definition 106
	Mark End 107
	Marking a Region From Here to Beginning of Buffer 107
	Marking a Region From Here to End of Buffer 107
	Marking Files for Deletion in Dired 170
	Marking Files to Be Hardcopied in Dired 172
Region	Marking Mode Variable 270

- Marking text 106
- Mark Page 107
- Mark Paragraph 107
- Mark region 67
- Commands to Mark Regions 106
- Overview of Commands to Mark Regions 106
- Commands to Mark Regions by Buffers 107
- Commands to Mark Regions by Lisp Expressions 106
- Commands to Mark Regions by Pages 107
- Commands to Mark Regions by Paragraphs 107
- Example of Commands to Mark Regions by Paragraphs 107
- Commands to Mark Regions by Words 106
- Mark Sexp 106
- Yank Matching 87
- Yank Pop Matching 88
- Example of Listing Variables by Matching a String 270
- Listing Variables by Matching a String 269
- Repeat Last Matching Minibuffer Command 88
- Insert Matching parentheses 233
- Current meaning of mouse clicks 67
- What the Lines Mean in the FED Character Box 301
- Character Select menu 293
- Dired Menu 167, 168
- Drawing Pane Menu 292
- Draw Mode Menu 292, 299, 331
- FED Character Select Menu 293
- FED Font Parameters Menu 293
- Font Parameters menu 293
- Gray Plane Menu 293
- Leaving Zmacs Via the System Menu 44
- Mousing on the FED Draw Mode Menu 331
- Mousing on the FED Font Parameters Menu 331
- Outside FED Command Menu 293
- Overview of the Editor Menu 57
- Selecting a FED Character From the Character Select Menu 298
- The Editor Menu 57
- The Zmacs Speller Menu 181
- Using the mouse in the draw mode menu 331
- Using the mouse in the Font Parameters menu 331
- FED Menu and Keyboard Commands 327
- Editor Menu Commands 57
- [Add in Gray] Font Editor menu item 304, 328
- [Center View] Font Editor menu item 323, 327
- [Clear Gray] Font Editor menu item 303, 328
- [Clear Points] Font Editor draw mode menu item 299
- [Configure] Font Editor menu item 291, 324, 327
- [Draw Line] Font Editor menu item 312, 327
- [Draw Spline] Font Editor menu item 312, 327
- [Edit Font] Font Editor menu item 297, 329
- [Erase All] Font Editor menu item 303, 309, 327
- [EXIT] Font Editor menu item 329
- [Flip Points] Font Editor draw mode menu item 299
- [Gray Char] Font Editor menu item 303, 328
- [Grid Size] Font Editor menu item 324, 327
- [HELP] Font Editor menu item 329
- [List Fonts] Font Editor menu item 297, 329
- [Move Black] Font Editor menu item 302, 327
- [Move Gray] Font Editor menu item 303, 328
- [Move View] Font Editor menu item 323, 327
- [Read File] Font Editor menu item 326, 329
- [Reflect] Font Editor menu item 309, 327

- [Rename Char] Font Editor menu item 298, 329
- [Rotate] Font Editor menu item 309, 327
- [Save Char] Font Editor menu item 298, 329
- [Set Points] Font Editor draw mode menu item 299
- [Set Sample] Font Editor menu item 321, 329
- [Show Font] Font Editor menu item 291, 297, 298, 329
- [Stretch] Font Editor menu item 327
- [Swap Gray] Font Editor menu item 303, 328
- Using the mouse with [List Fonts] Font Editor menu item 332
- Using the mouse with [Show Font] Font Editor menu item 298, 332
- [Write File] Font Editor menu item 326, 329
- FED Gray Plane Menu Items 328
- FED Outside World Interface Menu Items 329
- Sh-R:System menu mouse click 67
- Command menus 292
- FED Menus 292
- Using the mouse with Font Editor menus 292
- Source Compare Merge 142
- Source Compare Merge Installed Definition 145
- Source Compare Merge (m-X) Zmacs command 142
- Source Compare Merge Newest Definition 144
- Source Compare Merge Saved Definition 144
- Kill Merging 89
- Merging Characters with the FED Gray Plane 304
- Merging lines 24
- Using the META key while drawing characters 299
- Method for Searching for Appropriate Zmacs Commands 53
- Zmacs Midas Mode 177
- Minibuffer 20
- Echo Area's Minibuffer 20
- Evaluate Minibuffer 230
- More on the Minibuffer 59
- RETURN key in the minibuffer 59
- Yanking in the Minibuffer 59
- Repeat Last Minibuffer Command 59, 88
- Repeat Last Matching Minibuffer Command 88
- Getting Out of Minibuffer Prompts 48
- Minibuffer Prompts 48
- Minibuffer Response Format 59
- Minibuffer Response Help 59
- Minibuffer Responses 59
- More Ways to Enter Minor Mode 253
- Example of Filling Text with Auto Fill Mode Line's *Minor-mode* 21
- Built-in Customization Using Zmacs Minor Modes 253
- Definition of Zmacs Minor Modes 253
- Summary of Zmacs Minor Modes 254
- How Zmacs Minor Modes Work 253
- Minor version number 242
- Mirror imaging characters 309
- Misspelling 180
- M:Mark thing mouse click 67
- Mode 160, 254
- Auto Fill mode 256
- Default major Mode 253
- Example of Filling Text with Auto Fill Minor Mode 274
- Init File Form: Auto Fill in Text Mode 273
- Init File Form: Electric Shift Lock in Lisp Mode 273
- Init File Form: Setting Default Major Mode 176
- Overview of Setting the Major Mode 175
- Setting the Zmacs Major mode 146
- Two window

- Word Abbrev Mode 203
- Zmacs Bolio Mode 177
- Zmacs Electric P11 Mode 177
- Zmacs Fortran Mode 177
- Zmacs Fundamental Mode 176
- Zmacs Lisp Mode 176
- Zmacs Macsyma Mode 177
- Zmacs Midas Mode 177
- Zmacs P11 Mode 177
- Zmacs Teco Mode 177
- Zmacs Text Mode 176
- Creating a Fundamental Mode Buffer 139
- Setting Mode Hooks in Init Files 273
- Dired Mode in Zmacs 163
- Zmacs Mode Line 20
- Mode Line Example 22
- Mode Line's *Buffer* 21
- Mode Line's *Buffer-status* 22
- Mode Line's *Major-mode* 21
- Mode Line's *Minor-mode* 21
- Mode Line's *Position-flag* 22
- Mode Line's *Version* 22
- Find File In Fundamental Mode (m-X) Zmacs command 139
- Lisp Mode (m-X) Zmacs command 224
- Draw Mode Menu 292, 299, 331
- Mousing on the FED Draw Mode Menu 331
- Using the mouse in the draw mode menu 331
- [Clear Points] Font Editor draw mode menu item 299
- [Flip Points] Font Editor draw mode menu item 299
- [Set Points] Font Editor draw mode menu item 299
- Built-in Customization Using Zmacs Minor Modes 253
- Definition of Zmacs Minor Modes 253
- File Types and Zmacs Major Modes 256
- Summary of Zmacs Minor Modes 254
- User-Defined Zmacs Major Modes 256
- Zmacs Major Modes 256
- Zmacs Major Editing Modes 176
- How Zmacs Minor Modes Work 253
- Region Marking Mode Variable 270
- Region Right Margin Mode Variable 270
- c-X c-A Add Mode Word Abbrev 200
- Modification flag 130
- Modified Two Windows 146
- More HELP Commands for Finding Out About Zmacs Commands 53
- More on the Minibuffer 59
- More Ways to Enter Minibuffer Responses 59
- Motion 67
- Motion Along One Nesting Level 76
- Motion Among Top-Level Expressions 77
- Motion by Character 71
- Motion by Line 74
- Motion by Lisp Expression 76
- Motion by Lisp Expression 76, 93
- Motion by Page 80
- Motion by Page 80
- Motion by Paragraph 79
- Motion by Paragraph 79
- Motion by Sentence 73
- Motion by Word 72
- Motion Commands 71
- Description of
- Introduction to
- Introduction to

- Example of Negative Numeric Arguments with Motion Commands 71
- Example of Numeric Arguments with Motion Commands 71
- Goal Column and the Motion Commands 74
- Introduction to the Motion Commands 71
- Negative Numeric Arguments and Motion Commands 71
- Numeric Arguments and the Motion Commands 71
- Motion up and Down Nesting Levels 77
- Motion with Respect to the Whole Buffer 81
- Mouse 12, 67, 134, 142
- Creating a Region with the Mouse 101
- Drawing Characters in FED with the Mouse 299
- Drawing characters with the mouse 299
- Entering Zmacs with the Mouse 12
- Introduction to Using the Mouse 67
- Moving the Cursor with the Mouse 67
- Mouse as a graphic input device 291
- mouse click 67
- L:Move point mouse click 67
- M:Mark thing mouse click 67
- R:Menu mouse click 67
- Sh-2:Move to point mouse click 67
- Sh-M:Save/Kill/Yank mouse click 67
- Sh-R:System menu mouse click 67
- Current meaning of mouse clicks 67
- Mouse Documentation Line 67
- Mouse Documentation Line in Zmacs 67
- Using the mouse in the character select pane 331
- Using the mouse in the drawing pane 330
- Using the mouse in the draw mode menu 331
- Using the mouse in the Font Parameters menu 331
- Using the mouse in the register pane 331
- Using the mouse in the sample pane 331
- Installing a Mouse Macro 262
- Using the Mouse on History Elements 87
- Using the mouse on the character box 302
- Mouse-sensitive Zmacs commands 67
- Mouse Sensitivities in FED 330
- Using the mouse to enter Zmacs 12
- Using the mouse with Font Editor menus 292
- Using the mouse with List Files 148
- Using the mouse with [List Fonts] Font Editor menu item 332
- Using the mouse with [Show Font] Font Editor menu item 298, 332
- Mousing on the FED Character Select Pane 331
- Mousing on the FED Drawing Pane 330
- Mousing on the FED Draw Mode Menu 331
- Mousing on the FED Font Parameters Menu 331
- Mousing on the FED List Fonts and Show Font Displays 332
- Mousing on the FED Register Pane 331
- Mousing on the FED Sample Pane 331
- [Move Black] Font Editor menu item 302, 327
- Move cursor to beginning of line 66
- Move drawing in the gray plane 303, 328
- [Move Gray] Font Editor menu item 303, 328
- Summary of Cursor Movement 64
- Cursor movement commands 28, 71
- Move Over) 78
- Dired move point 168
- Move to Default Previous Point 102
- Move to Previous Point 102
- [Move View] Font Editor menu item 323, 327

Moving Around in Dired 168
 Moving Down to Lisp Comment on Next Line in
 Zmacs 227
 Moving Rest of Line Down in Zmacs 221
 Moving the cursor 67
 Description of Moving the Cursor 28
 Introduction to Moving the Cursor 28
 Overview of Moving the Cursor 64
 Summary of Moving the Cursor 28
 Moving the Cursor in Zmacs 63
 Moving the Cursor with the Mouse 67
 Moving the drawing 312, 327
 Moving the Drawing Horizontally and/or Vertically in
 FED 323
 Moving the Drawing in FED 312
 Moving the point 67
 Moving to a Specified Line 66
 Moving to end of buffer 81
 Saving and Moving to Locations in Registers 104
 Moving to Previous Points 102
 Moving up to Lisp Comment on Previous Line in
 Zmacs 227
 Multiple buffers 130
 Multiple Edit Callers Intersection 240
 Deleting Multiple File Versions in Dired 170
 Querying While Making Multiple Global Replacements in Zmacs 119
 Multiple List Callers Intersection 241
 Multiple text environment 36
 Deleting Multiple Versions 153
 Multiple windows 130

N**N****N**

SELECT N 44
 Function-Specs-to-Edit- n* buffer 13
 Name Last Kbd Macro (m-X) Zmacs command 261
 Set Visited File Name (m-X) Zmacs command 139
 Command Names 6
 Zmacs Buffer and File Names 130
 Names of commands 7, 52
 Create Spell Dictionary From Namespace Command 190
 Naming a File 138
 Naming a Keyboard Macro 261
 Negative Numeric Arguments and Motion
 Commands 71
 Negative Numeric Arguments with Motion
 Commands 71
 Example of Nesting Level 76
 Motion Along One Nesting Levels 77
 Motion up and Down New Buffer 137
 Reading a File Into a New Character 298
 Selecting a FED Character by Creating a new characters 298
 Creating Source Compare Newest Definition 144
 Source Compare Merge Newest Definition 144
 Buffer Flags for New Files 131
 Init File Form: Setting Find File Not to Create New Files 273
 Creating a New Font in FED 297
 Creating new fonts 297
 Starting a New Indented Lisp Comment Line in Zmacs 227
 New Line 24
 Newline characters 24

- Indenting New Line in Zmacs 219
 - Newlines 24
 - New Line with This Indentation in Zmacs 220
 - Creating New Zmacs Commands with Keyboard Macros 257
 - Append Next Kill 89
 - Moving Down to Lisp Comment on Next Line in Zmacs 227
 - Next Page 28, 80
 - Next Possibility 126
 - Displaying the Next Possibility 126
 - Example of Displaying the Next Possibility 127
 - Next Screen 28
 - Displaying the Next Screen 65
 - Dired Next Undumped 172
 - Nofill Attribute 160
 - Nofill file attribute 160
 - Set Nofill (m-X) Zmacs command 160
 - Nonmouse cursor 299, 300, 330
 - The FED Nonmouse Cursor 300
 - Dired Complement No Reap Flag 171
 - Example 1 of Zmacs Notation Conventions 9
 - Example 2 of Zmacs Notation Conventions 9
 - Example 3 of Zmacs Notation Conventions 9
 - Zmacs Manual Notation Conventions 9
 - Zmacs Notation Conventions and Examples 9
 - Finding Files That Have Not Been Backed up in Dired 172
 - note-private-patch function 249
 - @ note text formatting command 39
 - Entering Notifications 44
 - Init File Form: Setting Find File Not to Create New Files 273
 - Nullifying prefixes 48
 - Minor version number 242
 - Quadruple Numeric Arg 26
 - Numeric arguments 24, 26, 52
 - Defaults to Numeric Arguments 26
 - Example of Numeric Arguments 26
 - Overview of Numeric Arguments 26
 - Negative Numeric Arguments and Motion Commands 71
 - Numeric Arguments and the Motion Commands 71
 - Example of Numeric Arguments with Motion Commands 71
 - Example of Negative Numeric Arguments with Motion Commands 71
-
- c-X 0 Zmacs command 147
 - Count Occurrences 62
 - Zmacs Speller Accept Once command 181
 - Motion Along One Nesting Level 76
 - Change One Style Region 210
 - One Window 147
 - Returning to One Window 147
 - One Window Default Variable 270
 - Online documentation for commands 52
 - Online Documentation for Dired 167
 - Online documentation for prefixes 52
 - Operation 122
 - Example of a Tag Tables Replacement Operations in Zmacs 120
 - Other Types of Replacement Operations with Tag Tables 123
 - Performing Ordering Buffer Lists 272
 - Init File Form: Organization of the Screen 18
 - Introduction to the Organization of the Screen 18
 - Organization of the Zmacs Manual 4

Commands 148

P**P****P**

	SELECT	P 44
		P Dired command 172
Init File Form: Putting Buffers Into Current	Package	272
Setting the	Package	156
Set	Package (m-X) Zmacs command	156
	Packages	156
	Backward	Page 80
	Count Lines	Page 55
	Forward	Page 80
Introduction to Motion by	Page	80
Mark	Page	107
Motion by	Page	80
Next	Page	28, 80
Previous	Page	28, 80
Commands to Mark Regions by	Pages	107
Count	Pages	61
Black	pane	303
Drawing	pane	291, 330
FED Drawing	Pane	291
FED Prompt	Pane	292
FED Register	Pane	294
FED Sample	Pane	291
FED Status	Pane	293
Height and width of the drawing	pane	324
Mousing on the FED Character Select	Pane	331
Mousing on the FED Drawing	Pane	330
Mousing on the FED Register	Pane	331
Mousing on the FED Sample	Pane	331
Prompt	pane	292
Register	pane	294, 307, 331
Sample	pane	291, 299, 321, 331
Setting the Box Size in the FED Drawing	Pane	324
Setting the Height and Width of the FED Drawing	Pane	324
Size of boxes in the drawing	pane	324, 327
Status	pane	293
Using the mouse in the character select	pane	331
Using the mouse in the drawing	pane	330
Using the mouse in the register	pane	331
Using the mouse in the sample	pane	331
Drawing	Pane Menu	292
Backward	Paragraph	28, 79
Fill	Paragraph	110
Forward	Paragraph	28, 79
Introduction to Motion by	Paragraph	79
Mark	Paragraph	107
Motion by	Paragraph	79
Commands to Mark Regions by	Paragraphs	107
Count	Paragraphs	61
Example of Commands to Mark Regions by	Paragraphs	107
FED Font	Parameters Menu	293
Font	Parameters menu	293
Mousing on the FED Font	Parameters Menu	331
Using the mouse in the Font	Parameters menu	331
Find Unbalanced	Parentheses	56
Insert Matching	parentheses	233
Find Unbalanced	Parentheses (m-X) Zmacs command	56
Check Unbalanced	Parentheses When Saving Variable	271

- Current patch 246
 - In-progress patch 246
 - Send mail about patch 247
 - Add Patch Changed Definitions (m-X) 245
 - Add Patch Changed Definitions of Buffer (m-X) 245
 - Active patches 242, 246
 - Inactive patches 246
 - Making Patches 242
 - Show Patches (m-X) 246
 - Add region to patch file 242
 - Install patch file 247
 - Patch-File Attribute 161
 - Patch-File file attribute 161
 - Set Patch File (m-X) Zmacs command 161
 - Patching Programs in Zmacs 242
 - Abort Patch (m-X) 248
 - Add Patch (m-X) 244
 - Finish Patch (m-X) 247
 - Recompile Patch (m-X) 248
 - Reload Patch (m-X) 248
 - Resume Patch (m-X) 248
 - Select Patch (m-X) 246
 - Start Patch (m-X) 243
 - Start Private Patch (m-X) 244
 - Initial patch state 246
 - In-progress patch state 246
 - Default Pathnames in Dired 166
 - Entering Peek 44
- Saving Characters and Performing Operations with Tag Tables 123
 - Pieces of Characters in FED Registers 307
 - Pixel 254, 291
 - Zmacs Pl1 Mode 177
 - Zmacs Electric Pl1 Mode 177
 - Clear gray plane 303, 328
 - Getting Things Into the FED Gray Plane 303
 - Gray plane 293, 303
 - Merging Characters with the FED Gray Plane 304
 - Move drawing in the gray plane 303, 328
 - The FED Gray Plane 303
 - Gray Plane Menu 293
 - FED Gray Plane Menu Items 328
 - Retrieving the Black Plane While Manipulating FED Registers 307
 - c-X p1us-SIGN Add Global Word Abbrev 200
- Describe Variable At Point 54
 - Dired move point 55
 - Editor Window's Cursor and Point 168
 - Move to Default Previous Point 18
 - Move to Previous Point 102
 - Move to Previous Point 102
 - Moving the point 67
 - Exclamation point (!) line continuation indicator 25, 64
 - Swap Point And Mark 103
 - Point and the Region 100
 - Buffer pointers 100
 - L:Move point mouse click 67
 - Sh-2:Move to point mouse click 67
 - The Point-pdl 101
 - Clear all points 327
 - Moving to Previous Points 102
 - [Clear Points] Font Editor draw mode menu item 299

- [Flip Points] Font Editor draw mode menu item 299
- [Set Points] Font Editor draw mode menu item 299
- Yank Pop 59, 88
- Set Pop Mark 102
- Yank Pop Matching 88
- Default column position 75
- Mode Line's *Position-flag* 22
- Positioning the Drawing in FED 323
- Positioning the Window Around a Definition 66
- Possibility 126
- Example of Displaying the Next Possibility 127
- Next Possibility 126
- Possibility Buffers 126
- Set Fill Prefix 110
- Prefix character commands 7
- Prefix Command Does 52
- Prefix Commands 52
- Prefixes 48
- Getting Out of Keystroke Prefixes 48
- Nullifying prefixes 48
- Online documentation for prefixes 52
- Getting Out of Prefixes and Prompts 48
- Appending, Prepending, and Inserting Text in Zmacs 141
- Prepending a Region to a File 141
- Prepend To File (m-X) Zmacs command 141
- Previous Buffer 133
- Select Default Previous Buffer 133
- Displaying previous keystrokes 53
- Moving up to Lisp Comment on Previous Line in Zmacs 227
- Deleting the Previous Lisp Expression 93
- Move to Previous Page 28, 80
- Move to Default Previous Point 102
- Moving to Previous Point 102
- Previous Points 102
- Previous Screen 28
- Displaying the Previous Screen 66
- Deleting the Previous Sentence 97
- Deleting the Previous Word 92
- Start Private Patch (m-X) 244
- Procedure for Creating Zmacs Commands with Keyboard Macros 257
- Producing Formatted Text 35
- Compiling Lisp Programs in Zmacs 230
- Editing Lisp Programs in Zmacs 223
- Evaluating and Compiling Lisp Programs in Zmacs 229
- Evaluating Lisp Programs in Zmacs 229
- Introduction to Editing Lisp Programs in Zmacs 224
- Overview of Evaluating and Compiling Lisp Programs in Zmacs 229
- Patching Programs in Zmacs 242
- Zmacs Speller Prompt command 181
- FED Prompt pane 292
- Escaping from Prompt Pane 292
- Getting Out of Minibuffer prompts 48
- Getting Out of Prefixes and Prompts 48
- Minibuffer Prompts 48
- File Properties 150
- Show File Properties 150
- View File Properties 150
- Changing File Properties in Dired 168
- Change File Properties (m-X) Zmacs command 150

Show File Properties (m-X) Zmacs command 150
 Changing the Properties of a File 150
 Showing the Properties of a File 150
 Protecting Files From Being Deleted in Dired 171
 Protecting Files From Being Reaped in Dired 171
 @ p text environment 36
 Init File Form: Putting Buffers Into Current Package 272

Q

Q

Q

Q Dired command 167
 Q Font Editor command 329
 c-X Q Zmacs command 260
 Quadruple Numeric Arg 26
 Kbd Macro Query 260
 Update Attribute List Query 159
 Querying While Making Global Replacements in
 Zmacs 118
 Querying While Making Multiple Global Replacements
 in Zmacs 119
 Atom Query Replace 118
 Query Replace 121
 Query Replace Last Kill 120
 Query Replace LET Binding 121
 Query Replace (m-X) Zmacs command 118
 Quick Arglist 55
 Quotation text environment 36

R

R

R

R Dired command 169
 [Read File] Font Editor menu item 326, 329
 Reading a File Into a New Buffer 137
 Reading a File Into an Existing Buffer 137
 Reading and Writing FED Files 325
 Reading FED Files 325
 Reading font files 326
 m-X Read Spell Dictionary 190
 zwel: **read-spell-dictionary** function 193
 zwel: **read-standard-spell-dictionaries** function 194
 Read Word Abbrev File 202
 Down Real Line 28, 74, 95
 Up Real Line 28, 74, 95
 Init File Form: Setting Goal Column for
 Protecting Files From Being Reaped in Dired 171
 Dired Complement No Reap File (m-X) Zmacs command 153
 Reap Flag 171
 Recentering the Window 65
 Comparing Recent Versions of Files in Dired 169
 Error Recompile Patch (m-X) 248
 Introduction to recovery 48
 Redisplaying the Window 65
 Redisplaying the Window 65
 [Reflect] Font Editor menu item 309, 327
 Reflecting Drawings in FED 309
 Reflecting the drawing 309, 327
 REFRESH Font Editor command 330
 Region 100
 c-X c-J Change Style Region 210
 Change One Style Region 210

- Compiling a Region 109
- Count Lines Region 55
- Creating a Region 101
- Deleting a Region 109
- Evaluate Region 230
- Fill Region 110
- Filling a Region 109
- Format Region 42
- Hardcopying a Region 109
- Kill Region 30
- m-X Spell Region 184
- Make region 67
- Mark region 67
- Mark and the Region 100
- Point and the Region 100
- Save Region 109
- Saving a Region 109
- Two Windows Showing Region 146
- What is a Zmacs Region? 100
- Marking a Region From Here to Beginning of Buffer 107
- Marking a Region From Here to End of Buffer 107
- Creating Two Windows with the Region in Top 146
- Indenting Region in Zmacs 219
- Compile Region (m-X) Zmacs command 109
- Evaluate Region (m-X) Zmacs command 230
- Format Region (m-X) Zmacs command 35, 42
- Region-Manipulating Commands 109
- Region Marking Mode Variable 270
- Other Region-related Commands 110
- Region Right Margin Mode Variable 270
- Commands to Mark Regions 106
- Exchange Regions 109
- Getting Information About Buffers and Regions 61
- Introduction to Regions 100
- Overview of Commands to Mark Regions 106
- Transposing Regions 109
- Commands to Mark Regions by Buffers 107
- Commands to Mark Regions by Lisp Expressions 106
- Commands to Mark Regions by Pages 107
- Commands to Mark Regions by Paragraphs 107
- Example of Commands to Mark Regions by Paragraphs 107
- Commands to Mark Regions by Words 106
- Saving and Inserting Regions in Registers 104
- Changing Case of Regions in Zmacs 214
- Inserting and Removing Lisp Comments From Regions in Zmacs 228
- Working with Regions in Zmacs 99
- Appending a Region to a Buffer 141
- Appending a Region to a File 141
- Prepending a Region to a File 141
- Add region to patch file 242
- Indenting Region Uniformly in Zmacs 220
- Evaluate Region Verbose 230
- Creating a Region with Keystrokes 101
- Creating a Region with the Mouse 101
- Retrieving the Contents of a FED Register 307
- Saving a Drawing Into a FED Register 307
- Insert text from register into buffer 105
- Register pane 294, 307, 331
- FED Register Pane 294
- Mousing on the FED Register Pane 331
- Using the mouse in the register pane 331

- Retrieving the Black Plane While Manipulating FED Registers 294
- Saving and Inserting Regions in FED Registers 307
- Saving and Moving to Locations in FED Registers 104
- Saving Characters and Pieces of Characters in FED Registers 104
- Registers 307
- Registers in Zmacs 104
- Reindenting Expression in Zmacs 219
- Reinitializing Zmacs 12
- Reload Patch (m-X) 248
- Relocate cursor 67
- Inserting and Removing Lisp Comments From Regions in Zmacs 228
- Rename Buffer (m-X) Zmacs command 136
- [Rename Char] Font Editor menu item 298, 329
- Rename File 151
- Rename File (m-X) Zmacs command 151
- Renaming a File 151
- Selecting a FED Character by Renaming Characters 298
- Copying and Renaming Files 169
- Renaming Files in Dired 169
- Renaming the Buffer 136
- Reparsing Attribute List (m-X) Zmacs command 155
- Repeat Last Matching Minibuffer Command 88
- Repeat Last Minibuffer Command 59, 88
- Atom Query Replace 121
- Query Replace 118
- Evaluate and Replace Into Buffer 120
- Evaluate and Replace Into Buffer (m-X) Zmacs command 230
- Query Replace Last Kill 120
- Query Replace LET Binding 121
- Query Replace (m-X) Zmacs command 118
- Example of a Tag Tables Replacement Operation 122
- Other Types of Replacement Operations in Zmacs 120
- Making Global Replacements in Zmacs 118
- Querying While Making Global Replacements in Zmacs 118
- Querying While Making Multiple Global Replacements in Zmacs 119
- Replacing, and Sorting in Zmacs 113
- Replacing Strings Automatically 118
- Replacing Strings Automatically in Zmacs 118
- Reposition Window 66
- Re-reading a File Into the Buffer 138
- Searching, Respect to the Whole Buffer 81
- Overview of Locating and Locating and response 48
- Motion with Cancel Response Format 59
- Minibuffer Response Help 59
- Minibuffer Responses 59
- More Ways to Enter Minibuffer Rest of Line Down in Zmacs 221
- Moving Restoring text 49
- Setting Generation Resume Patch (m-X) 248
- Retention Count on Files in Dired 171
- Retrieving History Elements 87
- Retrieving the Black Plane While Manipulating FED Registers 307
- Retrieving the Contents of a FED Register 307
- Carriage return 7
- RETURN completion command 15
- RETURN key in the minibuffer 59
- Returning to One Window 147
- Zmacs Reverse Incremental Search 115
- Dired Reverse Undelete 170

- Revert Buffer (m-X) Zmacs command 138
- Reverting buffers 138, 139
- right command 53
- Finding the Right Edges of the FED Character Box 301
- Left and Right margin 254
- Region Right Margin Mode Variable 270
- R:Menu mouse click 67
- [Rotate] Font Editor menu item 309, 327
- Rotating Drawings in FED 309
- Rotating the drawing 309, 327
- @ r text environment 36
- Rubout 30
- RUBOUT Dired command 170
- RUBOUT key 24
- RUBOUT Zmacs character 90
- RUBOUT Zmacs command 30, 49
- c-X RUBOUT Zmacs command 30, 97

S

S

S

- S Font Editor command 329
- c-X S Zmacs command 32
- [Set Sample] Font Editor menu item 321, 329
- Sample pane 291, 299, 321, 331
- FED Sample Pane 291
- Mousing on the FED Sample Pane 331
- Using the mouse in the sample pane 331
- Sample string 291, 321, 329
- The FED Sample String 321
- What Histories Save 84
- Save All Files (m-X) 136
- m-X Save All Spell Dictionaries 192
- Source Compare [Save Char] Font Editor menu item 298, 329
- Source Compare Merge Saved Definition 144
- Source Compare Merge Saved Definition 144
- Save File 32, 138
- Save File Zmacs command 34
- Save Region 109
- m-X Save Spell Dictionary 191
- Saving a Drawing Into a FED Register 307
- Saving a File 34
- Saving and Inserting Regions in Registers 104
- Saving and Moving to Locations in Registers 104
- Saving a Region 109
- Saving Buffers 136
- Creating and Saving Buffers and Files 32
- Description of Creating and Saving Buffers and Files 32
- Summary of Creating and Saving Buffers and Files 32
- Example 1 of Writing and Saving Characters and Pieces of Characters in FED
- Example 2 of Writing and Registers 307
- Writing and Saving Keyboard Macros 260
- Writing and Saving Keyboard Macros 260
- Writing and Saving Keyboard Macros 259
- Check Unbalanced Parentheses When Saving the Buffer Contents to the File 138
- Saving Variable 271
- Scale fraction 303
- SCL syntax 158
- Scope of the Zmacs Manual 4
- Displaying the Next Screen 65
- Displaying the Previous Screen 66
- Introduction to the Organization of the Screen 18

- Next Screen 28
- Organization of the Screen 18
- Previous Screen 28
- Split Screen 147
- Splitting the Screen 147
- SCROLL Zmacs command 28
- Scroll bar 323
- Scrolling the drawing 323
- Scrolling the Drawing Horizontally and/or Vertically in FED 323
- Scrolling the Other Window 147
- Scroll Other Window 147
- Zmacs Incremental Search 114
- Zmacs Reverse Incremental Search 115
- Zmacs String Search 116
- Introduction to Tag Tables and Search Domains 122
- Tag Tables and Search Domains in Zmacs 122
- Searching for Appropriate Commands 53
- Searching for Appropriate Zmacs Commands 52
- Method for Searching for Appropriate Zmacs Commands 53
- Searching in Zmacs 114
- Overview of Searching in Zmacs 114
- Searching, Replacing, and Sorting in Zmacs 113
- Example of a Search String for HELP A 53
- SELECT C 44
- SELECT D 44
- SELECT E 12, 44
- Entering Zmacs with SELECT E 12
- SELECT F 44
- SELECT I 44
- SELECT key 44
- Leaving Zmacs with the SELECT Key 44
- SELECT L 44
- SELECT M 44
- SELECT N 44
- SELECT P 44
- SELECT T 44
- SELECT X 44
- Using Two Windows, Select Bottom 146
- Select Buffer 32, 133
- Select Default Previous Buffer 133
- Selected buffer 132
- Selecting a Character in FED 297
- Selecting a FED Character by Creating a New Character 298
- Selecting a FED Character by Renaming Characters 298
- Selecting a FED Character From the Character Select Menu 298
- Selecting a FED Character From the [Show Font] Display 298
- Selecting a FED Character with the C Command 298
- Selecting a font 329
- Selecting a Font in FED 294
- Selecting, Listing, and Examining Zmacs Buffers 132
- Character Select menu 293
- FED Character Select Menu 293
- Selecting a FED Character From the Character Select Menu 298
- Mousing on the FED Character Select Pane 331
- Using the mouse in the character select pane 331
- Select Patch (m-X) 246

- Using Two Windows,
 - Select Previous Buffer 133
 - Select Top 146
 - Semicolon (;) comment indicator 226
 - Send mail about patch 247
 - zwel:** ***send-mail-about-patch*** 247
 - Sensitivities in FED 330
 - Sentence 28, 73
 - Backward Sentence 30
 - Backward Kill Sentence 97
 - Deleting the Current Sentence 97
 - Deleting the Previous Sentence 97
 - Forward Sentence 73
 - Kill Sentence 30, 97
 - Motion by Sentence 73
 - Description of Zmacs Sentence Delimiters 73, 97
 - Deleting Sentences 97
 - Introduction to Deleting Sentences 97
 - zwel:** ***set-attribute-update-list*** global variable 159
 - Set Backspace (m-X) Zmacs command 159
 - Set Base (m-X) Zmacs command 160
 - Set commands for file and buffer attributes 159
 - Other** Set Commands for File and Buffer Attributes 159
 - Set Default Character Style 212
 - Set Fill Column 254
 - Set Fill Prefix 110
 - Set Fonts (m-X) Zmacs command 160
 - Set Goal Column 75
 - Set Lowercase (m-X) Zmacs command 160
 - Set Nofill (m-X) Zmacs command 160
 - Set Package (m-X) Zmacs command 156
 - Set Patch File (m-X) Zmacs command 161
 - [Set Points] Font Editor draw mode menu item 299
 - Set Pop Mark 102
 - [Set Sample] Font Editor menu item 321, 329
 - Set Syntax [m-X] 158
 - Settable Zmacs Variables 270
 - Set Tab Width (m-X) Zmacs command 161
 - Setting/Popping the Mark 102
 - Setting Buffer Attributes 159
 - Init File Form:** Setting Default Major Mode 273
 - Init File Form:** Setting Editor Variables in Init Files 272
 - Init File Form:** Setting Find File Not to Create New Files 273
 - Setting Generation Retention Count on Files in Dired 171
 - Init File Form:** Setting Goal Column for Real Line Commands 273
 - Setting Key Bindings in Init Files 274
 - Setting Mode Hooks in Init Files 273
 - Settings 269
 - Settings for Lisp 33, 137, 176, 224
 - Setting the Box Size in the FED Drawing Pane 324
 - Setting the Height and Width of the FED Drawing Pane 324
 - Setting the Key 267
 - Setting the Lisp Comment Column in Zmacs 227
 - Overview of** Setting the Major Mode 176
 - Setting the Package 156
 - Setting the Syntax for Symbolics Common Lisp 158
 - Setting the Zmacs Major Mode 175
 - Setting Variables 270
 - Set Variable 271
 - Set Variable (m-X) Zmacs command 271
 - Set Visited File Name (m-X) Zmacs command 139

- Backward Sexp 76
 - Backward Kill Sexp 30, 93
 - Forward Sexp 76
 - Kill Sexp 30, 93
 - Mark Sexp 106
 - Exchange Sexp 93
- Sh-2:Move to point mouse click 67
 - Shift keys 7
- Init File Form: Electric
 - Shift Lock in Lisp Mode 273
 - Sh-M:Save/Kill/Yank mouse click 67
 - Show Buffer (m-X) Zmacs command 135
 - Show Character Styles 212
 - m-X Show Contents of Spell Dictionary 193
 - Show Directory 149
 - Show Directory (m-X) Zmacs command 149
 - Show Documentation 54
 - Show File (m-X) Zmacs command 150
 - Show File Properties 150
 - Show File Properties (m-X) Zmacs command 150
 - Selecting a FED Character From the
 - Mousing on the FED List Fonts and
 - [Show Font] Display 298
 - Show Font Displays 332
 - [Show Font] Font Editor menu item 291, 297, 298, 329
 - Using the mouse with
 - [Show Font] Font Editor menu item 298, 332
 - Showing a Buffer 135
 - Showing a File 150
 - Showing a Keyboard Macro 258
 - Showing Region 146
 - Showing the Mark 103
 - Showing the Properties of a File 150
 - Show Keyboard Macro (m-X) Zmacs command 258
 - Show Login Directory 149
 - Show Login Directory (m-X) Zmacs command 149
 - Show Patches (m-X) 246
 - m-X Show Spell Dictionaries 191
 - Sh-R:System menu mouse click 67
 - Site-specific Speller Dictionaries 189
 - Adding
 - List the last sixty commands 53
 - List the last sixty keystrokes 53
 - Changing Window
 - Size 146
 - [Grid
 - Size] Font Editor menu item 324, 327
 - Setting the Box
 - Size in the FED Drawing Pane 324
 - Size of boxes in the drawing pane 324, 327
 - List
 - Some Word Abbrevs 202
 - Using Keyboard Macros to
 - Zmacs
 - Sort 261
 - Sorting Commands 128
 - Sorting in Zmacs 128
 - Overview of
 - Sorting in Zmacs 128
 - Searching, Replacing, and
 - Sorting in Zmacs 113
 - Sort Via Keyboard Macros 261
 - source code 224
 - Introduction to Locating
 - Source Code in Zmacs 235
 - source code of a function 12
 - Editing the
 - Source Code to Edit in Zmacs 235
 - Locating
 - Source Compare 142, 148
 - Example of a
 - Source Compare 142
 - Source Compare Installed Definition 145
 - Source Compare (m-X) Zmacs command 142
 - Source Compare Merge 142
 - Source Compare Merge Installed Definition 145

- Source Compare Merge (m-X) Zmacs command 142
 - Source Compare Merge Newest Definition 144
 - Source Compare Merge Saved Definition 144
 - Source Compare Newest Definition 144
 - Source Compare Saved Definition 144
 - SPACE 15
 - Space 30
 - SPACE completion command 15
 - SPACE Dired command 168
 - SPACE Zmacs command 14
 - Space for Kill/Yank Commands 273
 - Space in Lisp Code 274
 - spacing 161
 - Specified Line 66
 - Specifying and Listing Tag Tables 122
 - Specifying Other Contents 146
 - Specify Zmacs Variable Settings 269
 - Spell 185
 - Spell Buffer 180, 185
 - Spell Dictionaries 192
 - Spell Dictionaries 191
 - Spell Dictionary 192
 - Spell Dictionary 190
 - Spell Dictionary 193
 - Spell Dictionary 192
 - Spell Dictionary 190
 - Spell Dictionary 191
 - Spell Dictionary 193
 - Spell Dictionary From Namespace Command 190
 - Speller 180
 - Speller 179
 - Speller Accept command 181
 - Speller Accept Once command 181
 - Speller Commands for Spelling 184
 - Speller Dictionaries 187
 - Speller Dictionaries 189
 - Speller Dictionaries 188
 - Speller Dictionaries 187
 - Speller dictionary 187
 - Speller Dictionary Commands 190
 - Speller dictionary file 187
 - Speller Dictionary Functions 193
 - Speller dictionary list 187
 - Speller Dictionary Management 187
 - Speller list of dictionaries 187
 - Speller Menu 181
 - Speller Prompt command 181
 - Spell File 185
 - Spelling 184
 - Spelling correction 180
 - Spell Region 184
 - (Spell This Word) 184
 - Spell Word 184
 - spline 327
 - Spline] Font Editor menu item 312, 327
 - Split Screen 147
 - Splitting the Screen 147
 - SQUARE Key 274
 - Srccom 169
 - Standard comtab 267
 - Standard TV Fonts 287
- Delete Horizontal
 - HELP
 - Init File Form: Fixing White
 - Init File Form: White
 - Vertical
 - Moving to a
 - Creating Two Windows,
 - How to
 - m-X Tags
 - m-X
 - m-X Save All
 - m-X Show
 - m-X Add Word to
 - m-X Compile
 - m-X Delete Word From
 - m-X Kill
 - m-X Read
 - m-X Save
 - m-X Show Contents of
 - Create
 - Using the Zmacs
 - Zmacs
 - Zmacs
 - Zmacs
 - Adding Site-specific
 - Adding User-specific
 - Introduction to
 - The Zmacs
 - Zmacs
 - m-X
 - Speller Commands for
 - m-X
 - m-\$
 - m-X
 - Draw a cubic
 - [Draw
 - Init File Form: c-m-L on the
 - Dired

- Getting Started in Zmacs 11
- Starting a Keyboard Macro 258
- Starting a New Line 24
- Starting Zmacs 12
- Start Kbd Macro 257
- Erase backward to
 - start of line 96
 - Start Patch (m-X) 243
 - Start Private Patch (m-X) 244
- Initial patch
 - state 246
- In-progress patch
 - state 246
- Zmacs Commands for Finding Out About the
 - State of Buffers 278
 - State of Zmacs 279
 - Status line documentation 323
 - Status pane 293
 - Status Pane 293
- FED
 - [Stretch] Font Editor menu item 327
 - Stretching a character 327
 - Stretching a Drawing Horizontally in FED 316
 - Stretching a Drawing Vertically in FED 316
 - Stretching and Contracting Drawings in FED 316
- String 270
- String 269
- string 291, 321, 329
- String 321
- String for HELP A 53
- String (m-X) Zmacs command 118
- String-matching in yank commands 85
- string-matching yank command 87
- Strings Automatically 118
- Strings Automatically in Zmacs 118
- String Search 116
- Style 211
- Style 212
- Style 212
- Style Character 210
- Style Commands 206
- Style Commands in Zmacs 210
- style for 206
- Style Region 210
- Style Region 210
- styles 210, 211, 212
- Styles 212
- Styles in Zmacs 205
- Style Word 211
- subheading 39
- Subheading text environment 36
- Subsystem 291
- Summary 165
- Summary 277
- Summary of Creating and Saving Buffers and Files 32
- Summary of Cursor Movement 64
- Summary of Erasing Text 30
- Summary of Moving the Cursor 28
- Summary of Zmacs Minor Modes 254
- Support Buffers 126
- Supported file formats 326
- [Swap Gray] 303
- [Swap Gray] Font Editor menu item 303, 328
- Swap Point And Mark 103
- Symbolics Common Lisp 158
- Example of Listing Variables by Matching a
 - Listing Variables by Matching a
 - Sample
 - The FED Sample
 - Example of a Search
 - Replace
 - c-sh-Y
- Overview of Locating and Replacing
 - Locating and Replacing
 - Zmacs
 - c-m-J Change Typein
 - Find Character in
 - Set Default Character
 - c-J Change
 - Introduction to the Character
 - Character
 - Character
 - c-X c-J Change
 - Change One
 - Character
 - Show Character
 - Using Character
 - m-J Change
 - Text
- FED, the
 - Dired Command
 - Zmacs Help Command
- Getting Things Into Gray with
- Setting the Syntax for

Default	syntax 158
SCL	syntax 158
Zetalisp	syntax 158
	Syntax attribute 157
Base and	Syntax Defaults 157
Base and	Syntax Default Settings for Lisp 33, 137, 176, 224
Setting the	Syntax for Symbolics Common Lisp 158
Set	Syntax (m-X) 158
Entering File	System Editor 44
Edit	System Files 238
Leaving Zmacs Via the	System Menu 44

T

T

T

SELECT	T 44
c-X	T Zmacs command 109
	TAB 217
	TAB in <code>zl:loop</code> macro 216
	@ tabclear text formatting command 39
	@ tabdivide text formatting command 39
Compile Changed Definitions of Tag	Table 231
Command	tables 7, 267
Introduction to Zmacs Command	Tables 7
Performing Operations with Tag	Tables 123
Specifying and Listing Tag	Tables 122
Introduction to Tag	Tables and Search Domains 122
Tag	Tables and Search Domains in Zmacs 122
Example of a Tag	Tables Replacement Operation 122
Example 1 of Making	Tables Using Keyboard Macros 264
Example 2 of Making	Tables Using Keyboard Macros 265
Making	Tables Using Keyboard Macros 263
How Tag	Tables Work 122
	Table text environment 36
	@ tabset text formatting command 39
Example of Using	Tabs to Format Text 41
	Tab-Width Attribute 161
	Tab-Width file attribute 161
	Set Tab Width (m-X) Zmacs command 161
m-X	Tags Spell 185
Compile Changed Definitions of	Tag Table 231
Performing Operations with	Tag Tables 123
Specifying and Listing	Tag Tables 122
Introduction to	Tag Tables and Search Domains 122
	Tag Tables and Search Domains in Zmacs 122
Example of a	Tag Tables Replacement Operation 122
How	Tag Tables Work 122
Zmacs	Teco Mode 177
zwel:	*temp-file-type-list* variable 153
Entering	Terminal 44
Deleting Vs. Killing	Text 84
Description of Erasing	Text 30
Erasing	text 90
Example of Using Tabs to Format	Text 41
Inserting	Text 24
Introduction to Erasing	Text 30
Introduction to Inserting	Text 24
Marking	text 106
Overview of Deleting Vs. Killing	Text 84
Producing Formatted	Text 35
Restoring	text 49
Summary of Erasing	Text 30

Transposing Lines of	Text	96
Zmacs Commands for Formatting	Text	35
Getting	Text Back	49
@b	text environment	36
@c	text environment	36
@g	text environment	36
@i	text environment	36
@p	text environment	36
@r	text environment	36
@t	text environment	36
Boldface	text environment	36
Center	text environment	36
Description	text environment	36
Display	text environment	36
Enumerate	text environment	36
Equation	text environment	36
Example	text environment	36
Figure	text environment	36
Flushleft	text environment	36
Flushright	text environment	36
Format	text environment	36
Fullpagefigure	text environment	36
Fullpagetable	text environment	36
Heading	text environment	36
Italics	text environment	36
Itemize	text environment	36
Majorheading	text environment	36
Multiple	text environment	36
Outputexample	text environment	36
Quotation	text environment	36
Subheading	text environment	36
Table	text environment	36
Verbatim	text environment	36
	Text example	39
	@	text formatting command 39
	@#	text formatting command 39
	@*	text formatting command 39
	@.	text formatting command 39
	@=	text formatting command 39
	@>	text formatting command 39
	@blankspace	text formatting command 39
	@caption	text formatting command 39
	@foot	text formatting command 39
	@note	text formatting command 39
	@tabclear	text formatting command 39
	@tabdivide	text formatting command 39
	@tabset	text formatting command 39
	@\	text formatting command 39
	@^	text formatting command 39
	Basic	Text Formatting Commands 39
	Basic	Text Formatting Environments 36
	Introduction to	Text Formatting in Zmacs 35
	Insert	text from register into buffer 105
		Text heading 39
Appending, Prepending, and Inserting	Text in Zmacs	141
Deleting and Transposing	Text in Zmacs	83
Init File Form: Auto Fill in	Text Mode	274
Zmacs	Text Mode	176
	Text subheading	39
Example of Filling	Text with Auto Fill Minor Mode	253
Finding Files	That Have Not Been Backed up in Dired	172

- How They Work 155
- M:Mark thing mouse click 67
- Getting Things Into Gray with [Gray Char] 303
- Getting Things Into Gray with [Swap Gray] 303
- Getting Things Into the FED Gray Plane 303
- New Line with This Indentation in Zmacs 220
- m-\$ (Spell This Word) 184
- Append To Buffer 141
- Append To File (m-X) Zmacs command 141
- Prepend To File (m-X) Zmacs command 141
- Toggle 253
- Creating Two Windows with the Region in Top 146
- Using Two Windows, Select Top 146
- Top Edge of the FED Character Box 301
- Abort At Top Level 48
- Motion Among Top-Level Expressions 77
- Trace 56
- FED Configuration and Drawing Trace (m-X) Zmacs command 56
- Transformation 327
- Transformations on Characters in FED 309
- Transposing Characters 90
- Transposing Characters 90
- Transposing Lines 95
- Introduction to Deleting and Transposing Lines 95
- Transposing Lines of Text 96
- Transposing Lisp Expressions 93
- Transposing Lisp Expressions 93
- Introduction to Deleting and Transposing Lisp Expressions 93
- Transposing Regions 109
- Deleting and Transposing Text in Zmacs 83
- Transposing Words 92
- Deleting and Transposing Words 92
- Introduction to Deleting and Transposing Words 92
- Getting Out of Trouble 48
- Overview of Getting Out of Trouble 48
- Attributes of TV Fonts 285
- Standard TV Fonts 287
- Two window mode 146
- Two Windows 146
- Modified Two Windows 146
- Using Two Windows 146
- View Two Windows 146
- Using Two Windows, Select Bottom 146
- Using Two Windows, Select Top 146
- Two Windows Showing Region 146
- Creating Two Windows, Specifying Other Contents 146
- Creating Two Windows with the Region in Top 146
- Finding Out What You Have Typed 53
- c-m-J Change Typein Style 211
- Editor Window's Typeout 18
- Typeout 18
- Typeout window 18, 51
- Canonical types 152
- File Types and Zmacs Major Modes 256
- Other Types of Replacement Operations in Zmacs 120
- Correcting Typical use of Zmacs 224
- Typos 24

U

U
 U Dired command 170
 c-X U Unexpand Last Word 203
 HELP U Zmacs command 14, 53
 Find Unbalanced Parentheses 56
 Find Unbalanced Parentheses (m-X) Zmacs command 56
 Check Unbalanced Parentheses When Saving Variable 271
 Dired Undelete 170
 Dired Reverse Undelete 170
 Dired Next Undo all changes to buffer 138
 c-X U Undumped 172
 Indenting Region Unexpand Last Word 203
 Motion Uniformly in Zmacs 220
 Finding Files That Have Not Been Backed Unknown attribute 156
 Backward up and Down Nesting Levels 77
 Forward Update Attribute List (m-X) Zmacs command 156
 Kill Backward Update Attribute List Query 159
 Moving Updating the Dired Display 164
 Adding up in Dired 172
 Up Line 74, 95
 up List 77
 up List 77
 Up List (c-m-X) Zmacs command 93
 Up Real Line 28, 74, 95
 up to Lisp Comment on Previous Line in Zmacs 227
 User-Defined Zmacs Major Modes 256
 User-specific Speller Dictionaries 188

U

U

V

V
 HELP V 54
 c-HELP V Dired command 169
 c-m-? V Font Editor command 321, 329
 c-X V Zmacs command 269
 HELP V Zmacs command 269
 V Zmacs command 135
 V Zmacs command 14, 54, 269
 Check Unbalanced Parentheses When Saving Variable 271
 Definition of a Zmacs Variable 269
 Describe Variable 269
 fs:*file-type-mode-allst* variable 256
 One Window Default Variable 270
 Region Marking Mode Variable 270
 Region Right Margin Mode Variable 270
 Set Variable 271
 zl:ibase global variable 160
 zwel:*file-versions-kept* variable 153
 zwel:*major-mode-translations* variable 256
 zwel:*set-attribute-update-list* global variable 159
 zwel:*temp-file-type-llst* variable 153
 Describe Variable Apropos 269
 Describe Variable Apropos Zmacs command 269
 Describe Variable At Point 55
 Describe Variable (m-X) Zmacs command 269
 Set Variable (m-X) Zmacs command 271
 Describing Zmacs Variables 269
 Finding Out About Zmacs Variables 269
 List Variables 269
 Listing Zmacs Variables 269
 Settable Zmacs Variables 270

V

V

- Setting Variables 270
- Zmacs variables 54
- Example of Listing Variables by Matching a String 270
- Listing Variables by Matching a String 269
- How to Specify Zmacs Variable Settings 269
- Setting Editor Variables in Init Files 272
- List Variables (m-X) Zmacs command 269
- Variable-width fonts 286
- Verbatim text environment 36
- Verbose 230
- Version 22
- version number 242
- Versions 145
- Versions 144
- Versions 144
- versions 169
- Versions 153
- versions 130
- Versions in Dired 170
- Versions of Files in Dired 169
- Vertically in FED 316
- Vertically in FED 323
- Vertically in FED 323
- Vertically in FED 316
- Vertical spacing 161
- Via Keyboard Macros 261
- Via the System Menu 44
- View Buffer 135
- View File 150
- View File 169
- View File Properties 150
- [View] Font Editor menu item 323, 327
- [Move View] Font Editor menu item 323, 327
- Viewing and Altering a Character in the FED
 - Character Box 301
- Viewing and Editing File Contents in Dired 169
- Viewing File Attributes in Dired 168
- Viewing the Editor Command History 86
- Viewing the Kill History 85
- View Two Windows 146
- Set Visited File Name (m-X) Zmacs command 139
- Visit File 137
- Deleting Vs. Killing Text 84
- Overview of Deleting Vs. Killing Text 84
- Vsp Attribute 161
- Vsp file attribute 161
- Set Vsp (m-X) Zmacs command 161

W

W

W

- c-X W Zmacs command 32
- HELP W Zmacs command 14
- HELP W Zmacs command 54
- Lisp Compiler Warnings 232
- Warnings about file attribute lists 156
- More Ways to Enter Minibuffer Responses 59
- Finding Out What an Extended Command Does 52
- Finding Out What a Prefix Command Does 52
- Example of Finding Out What a Zmacs Command Does 51
- Finding Out What a Zmacs Command Does 51
- What Histories Save 84

Forward Word 28, 72
 Kill Word 30, 92
 m-\$ (Spell This Word) 184
 m-J Change Style Word 211
 m-X Spell Word 184
 Motion by Word 72
 Word abbrev 198
 c-X c-A Add Mode Word Abbrev 200
 c-X plus-SIGN Add Global Word Abbrev 200
 Make Word Abbrev 202
 Read Word Abbrev File 202
 Write Word Abbrev File 203
 Word Abbreviation Commands 200
 Word Abbreviations 197
 Using Word Abbreviations 198
 Word Abbrev Mode 203
 Word abbrevs 198
 Edit Word Abbrevs 201
 Insert Word Abbrevs 201
 Kill All Word Abbrevs 201
 List Word Abbrevs 202
 List Some Word Abbrevs 202
 m-X Delete Word From Spell Dictionary 193
 Commands to Mark Regions by Words 106
 Count Words 61
 Deleting and Transposing Words 92
 Exchange Words 92
 Introduction to Deleting and Transposing Words 92
 Transposing Words 92
 Changing Case of Words in Zmacs 214
 m-X Add Word to Spell Dictionary 192
 How Tag Tables Work 122
 How They Work 155
 How Zmacs Keyboard Macros Work 257
 How Zmacs Minor Modes Work 253
 Working with Buffers and Files in Zmacs 130
 Working with Buffers and Files in Zmacs 130
 Working with Regions in Zmacs 99
 Overview of
 How Key Bindings Work: the Comtab 267
 FED Outside World Interface Menu Items 329
 Wraparound Lines 64
 Wraparound Lines in the Editor Window 64
 Wrapping Lines 25
 Write File 32, 138
 [Write File] Font Editor menu item 326, 329
 Write File Zmacs command 34
 Write Word Abbrev File 203
 Writing and Saving Keyboard Macros 259
 Example 1 of Writing and Saving Keyboard Macros 260
 Example 2 of Writing and Saving Keyboard Macros 260
 Writing FED Files 326
 Reading and Writing FED Files 325
 Writing font files 326
 Writing the Buffer Contents to a File 138

X

X

SELECT X 44

X

Y

Y

Yank 87
 yank command 15
 c- \emptyset c-m-Y yank command 15
 c-m-Y yank command 15
 c-0 c-Y yank command 15
 c-sh-Y string-matching yank command 87
 c-Y yank command 15, 87
 m-sh-Y yank command 88
 m-Y yank command 15, 59, 88
 String-matching in yank commands 85
 Yanking 15, 49
 Introduction to Yanking 15
 Yanking in the command history 15
 Yanking in the kill history 15
 Yanking in the Minibuffer 59
 Yank Matching 87
 Yank Pop 59, 88
 Yank Pop Matching 88
 Finding Out What You Have Typed 53

Y

Z

Z

Zetalisp syntax 158
 zl:base global variable 160
 How to Use the
 The
 TAB in
 Indentation in
 + flag in
 Aligning Indentation in
 Appending, Prepending, and Inserting Text in
 Buffer and File Attributes in
 Centering the Current Line in
 Changing Case and Indentation in
 Changing Case in
 Changing Case of Buffers in
 Changing Case of Regions in
 Changing Case of Words in
 Character Style Commands in
 Commenting Lisp Code in
 Comparing Files and Buffers in
 Compiling Lisp Programs in
 Creating a New Indented Lisp Comment Line in
 Deleting and Transposing Text in
 Deleting Blank Line in
 Deleting Indentation in
 Dired Mode in
 Editing Lisp Programs in
 Entering
 Evaluating and Compiling Lisp Programs in
 Evaluating Lisp Programs in
 Executing CP Commands From
 Expanding Lisp Expressions in
 Getting Help in
 Getting Started in

Zmacs 131
 Zmacs 220
 Zmacs 141
 Zmacs 155
 Zmacs 219
 Zmacs 213
 Zmacs 214
 Zmacs 214
 Zmacs 214
 Zmacs 214
 Zmacs 210
 Zmacs 226
 Zmacs 142
 Zmacs 230
 Zmacs 227
 Zmacs 83
 Zmacs 221
 Zmacs 220
 Zmacs 163
 Zmacs 223
 Zmacs 12, 44
 Zmacs 229
 Zmacs 229
 Zmacs 43
 Zmacs 234
 Zmacs 47
 Zmacs 11

Z

Going Back to First Indented Character in	Zmacs	219
Indentation in	Zmacs	216
Indenting Current Line in	Zmacs	216
Indenting for Lisp Comment in	Zmacs	226
Indenting New Line in	Zmacs	219
Indenting Region in	Zmacs	219
Indenting Region Uniformly in	Zmacs	220
Inserting and Removing Lisp Comments From Regions in	Zmacs	228
Inserting Blank Line in	Zmacs	221
Introduction to	Zmacs	6
Introduction to Customizing	Zmacs	252
Introduction to Editing Lisp Programs in	Zmacs	224
Introduction to Entering	Zmacs	12
Introduction to Locating Source Code in	Zmacs	235
Introduction to Text Formatting in	Zmacs	35
Invoking	Zmacs	12
Killing a Lisp Comment in	Zmacs	226
Leaving	Zmacs	44
Locating and Replacing Strings Automatically in	Zmacs	118
Locating Source Code to Edit in	Zmacs	235
Making Global Replacements in	Zmacs	118
Manipulating Buffers and Files in	Zmacs	129
Mouse Documentation Line in	Zmacs	67
Moving Down to Lisp Comment on Next Line in	Zmacs	227
Moving Rest of Line Down in	Zmacs	221
Moving the Cursor in	Zmacs	63
Moving up to Lisp Comment on Previous Line in	Zmacs	227
New Line with This Indentation in	Zmacs	220
Other Types of Replacement Operations in	Zmacs	120
Overview of	Zmacs	6
Overview of Changing Case in	Zmacs	214
Overview of Commenting Lisp Code in	Zmacs	226
Overview of Evaluating and Compiling Lisp Programs in	Zmacs	229
Overview of Indentation in	Zmacs	216
Overview of Leaving	Zmacs	44
Overview of Searching in	Zmacs	114
Overview of Sorting in	Zmacs	128
Overview of Working with Buffers and Files in	Zmacs	130
Parenthesizing Lisp Expressions in	Zmacs	233
Patching Programs in	Zmacs	242
Querying While Making Global Replacements in	Zmacs	118
Querying While Making Multiple Global Replacements in	Zmacs	119
Registers in	Zmacs	104
Reindenting Expression in	Zmacs	219
Reinitializing	Zmacs	12
Searching in	Zmacs	114
Searching, Replacing, and Sorting in	Zmacs	113
Setting the Lisp Comment Column in	Zmacs	227
Sorting in	Zmacs	128
Starting	Zmacs	12
Tag Tables and Search Domains in	Zmacs	122
Typical use of	Zmacs	224
Using Character Styles in	Zmacs	205
Using the mouse to enter	Zmacs	12
Working with Buffers and Files in	Zmacs	130
Working with Regions in	Zmacs	99
Zmacs Commands for Finding Out About the State of	Zmacs	279
	Zmacs	Bolio Mode 177

	Current	Zmacs Buffer	132
		Zmacs Buffer and File Names	130
		Zmacs Buffer Commands	133
		Zmacs Buffer History	132
Selecting, Listing, and Examining		Zmacs Buffers	132
	RUBOUT	Zmacs character	90
	ABORT	Zmacs command	48
Append To File (m-X)		Zmacs command	141
Arglist (m-X)		Zmacs command	55
	c-%	Zmacs command	118
	c-;	Zmacs command	226
	c-=	Zmacs command	54
	c-A	Zmacs command	28, 74, 95
	c-B	Zmacs command	28, 72
	c-D	Zmacs command	30, 49, 90
	c-E	Zmacs command	28, 74, 95
	c-F	Zmacs command	28, 72
	c-G	Zmacs command	48
	c-HELP V	Zmacs command	269
	c-K	Zmacs command	30, 95
	c-L	Zmacs command	65
	c-m-(Zmacs command	77
	c-m-)	Zmacs command	77
	c-m-;	Zmacs command	226
	c-m-? V	Zmacs command	269
	c-m-@	Zmacs command	106
	c-m-A	Zmacs command	78
	c-m-B	Zmacs command	76
	c-m-D	Zmacs command	77
	c-m-E	Zmacs command	78
	c-m-F	Zmacs command	76
	c-m-H	Zmacs command	106
	c-m-K	Zmacs command	30, 93
	c-m-L	Zmacs command	133
	c-m-N	Zmacs command	76
	c-m-O	Zmacs command	221
	c-m-P	Zmacs command	76
	c-m-Q	Zmacs command	219
	c-m-R	Zmacs command	66
	c-m-RUBOUT	Zmacs command	30, 93
	c-m-sh-E	Zmacs command	230
	c-m-SPACE	Zmacs command	102
	c-m-T	Zmacs command	93
	c-m-U	Zmacs command	77
	c-m-V	Zmacs command	147
	c-m-Z	Zmacs command	230
	c-m-[Zmacs command	78
	c-m-\	Zmacs command	219
	c-m-]	Zmacs command	78
	c-m-^	Zmacs command	220
	c-N	Zmacs command	28, 74, 95, 257
	c-O	Zmacs command	221
	c-P	Zmacs command	28, 74, 95
	c-sh-A	Zmacs command	55
	c-sh-C	Zmacs command	109
	c-sh-D	Zmacs command	55
	c-sh-E	Zmacs command	230
	c-sh-V	Zmacs command	55
	c-SPACE	Zmacs command	102
	c-T	Zmacs command	90
	c-U	Zmacs command	26

c-V	Zmacs command	28, 65
c-W	Zmacs command	30, 109
c-X	Zmacs command	257
c-X)	Zmacs command	257
c-X 1	Zmacs command	147
c-X 2	Zmacs command	146
c-X 3	Zmacs command	146
c-X 4	Zmacs command	146
c-X 8	Zmacs command	146
c-X ;	Zmacs command	227
c-X =	Zmacs command	54
c-X A	Zmacs command	141
c-X B	Zmacs command	32, 33, 133
c-X c-;	Zmacs command	228
c-X c-B	Zmacs command	131, 134
c-X c-D	Zmacs command	149
c-X c-F	Zmacs command	33, 34
c-X c-I	Zmacs command	220
c-X c-L	Zmacs command	214
c-X c-m-L	Zmacs command	133
c-X c-m-SPACE	Zmacs command	102
c-X c-N	Zmacs command	75
c-X c-O	Zmacs command	30, 221
c-X c-P	Zmacs command	107
c-X c-S	Zmacs command	34, 138
c-X c-T	Zmacs command	96
c-X c-U	Zmacs command	214
c-X c-V	Zmacs command	137
c-X c-W	Zmacs command	34, 138
c-X c-X	Zmacs command	103
c-X D	Zmacs command	163
c-X E	Zmacs command	258
c-X F	Zmacs command	32, 254
c-X L	Zmacs command	55
c-X O	Zmacs command	147
c-X Q	Zmacs command	260
c-X RUBOUT	Zmacs command	30, 97
c-X S	Zmacs command	32
c-X T	Zmacs command	109
c-X V	Zmacs command	135
c-X W	Zmacs command	32
c-X [Zmacs command	28, 80
c-X]	Zmacs command	28, 80
c-X ^	Zmacs command	146
Change File Properties (m-X)	Zmacs command	150
Clean Directory (m-X)	Zmacs command	153
CLEAR-INPUT	Zmacs command	96
Compile Region (m-X)	Zmacs command	109
Copy File (m-X)	Zmacs command	152
Create Directory (m-X)	Zmacs command	148
Create Link (m-X)	Zmacs command	152
Deinstall Macro (m-X)	Zmacs command	262
Delete File (m-X)	Zmacs command	153
Describe Variable (m-X)	Zmacs command	269
Dired (m-X)	Zmacs command	163
Edit Buffers (m-X)	Zmacs command	134
Edit Definition m-.	Zmacs Command	235
Edit Directory (m-X)	Zmacs command	163
End Kbd Macro	Zmacs command	257
Evaluate and Replace Into Buffer (m-X)	Zmacs command	230
Evaluate Buffer (m-X)	Zmacs command	230

Evaluate Changed Definitions (m-X)	Zmacs command 230
Evaluate Changed Definitions of Buffer (m-X)	Zmacs command 230
Evaluate Into Buffer (m-X)	Zmacs command 230
Evaluate Region (m-X)	Zmacs command 230
Execute Command Into Buffer (m-X)	Zmacs command 136
Find File	Zmacs command 33, 34
Find File In Fundamental Mode (m-X)	Zmacs command 139
Find Unbalanced Parentheses (m-X)	Zmacs command 56
Format Buffer (m-X)	Zmacs command 35, 42
Format File (m-X)	Zmacs command 35, 42
Format Region (m-X)	Zmacs command 35, 42
Hardcopy Buffer (m-X)	Zmacs command 136
Hardcopy File (m-X)	Zmacs command 151
HELP ?	Zmacs command 14
HELP A	Zmacs command 14, 53
HELP C	Zmacs command 14, 52
HELP D	Zmacs command 14, 52
HELP L	Zmacs command 14, 53
HELP SPACE	Zmacs command 14
HELP U	Zmacs command 14, 53
HELP V	Zmacs command 14, 54, 269
HELP W	Zmacs command 14
HELP W	Zmacs command 54
Insert Buffer (m-X)	Zmacs command 141
Insert File (m-X)	Zmacs command 141
Install Command (m-X)	Zmacs command 268
Install Macro (m-X)	Zmacs command 262
Kill Backward Up List (c-m-X)	Zmacs command 93
LINE	Zmacs command 219
Lisp Mode (m-X)	Zmacs command 224
List Buffers	Zmacs command 131
List Changed Definitions of Buffer (m-X)	Zmacs command 239
List Definitions (m-X)	Zmacs command 239
List Files (m-X)	Zmacs command 148
List Fonts (m-X)	Zmacs command 287
List Variables (m-X)	Zmacs command 269
m-%	Zmacs command 118
m-)	Zmacs command 78
m-;	Zmacs command 226
m-<	Zmacs command 28, 81
m-=	Zmacs command 55
m->	Zmacs command 28, 81
m-@	Zmacs command 106
m-A	Zmacs command 28
m-B	Zmacs command 28, 72
m-C	Zmacs command 214
m-D	Zmacs command 30, 92
m-E	Zmacs command 73
m-ESCAPE	Zmacs command 230
m-F	Zmacs command 28, 72
m-H	Zmacs command 107
m-K	Zmacs command 30, 97
m-L	Zmacs command 214
m-LINE	Zmacs command 227
m-N	Zmacs command 227
m-O	Zmacs command 220
m-P	Zmacs command 227
m-R	Zmacs command 66
m-RUBOUT	Zmacs command 30, 92
m-S	Zmacs command 219
m-SCROLL	Zmacs command 28, 66

	m-sh-D	Zmacs command	54
	m-sh-E	Zmacs command	230
	m-T	Zmacs command	92
	m-U	Zmacs command	214
	m-V	Zmacs command	28, 66
	m-W	Zmacs command	109
	m-Z	Zmacs command	231
	m-[Zmacs command	28, 79
	m-\	Zmacs command	30
	m-]	Zmacs command	28, 79
	m-~	Zmacs command	30, 220
Name Last Kbd Macro (m-X)		Zmacs command	261
Prepend To File (m-X)		Zmacs command	141
Query Replace (m-X)		Zmacs command	118
Reap File (m-X)		Zmacs command	153
Rename Buffer (m-X)		Zmacs command	136
Rename File (m-X)		Zmacs command	151
Repars Attribute List (m-X)		Zmacs command	155
Replace String (m-X)		Zmacs command	118
Revert Buffer (m-X)		Zmacs command	138
RUBOUT		Zmacs command	30, 49
Save File		Zmacs command	34
SCROLL		Zmacs command	28
Set Backspace (m-X)		Zmacs command	159
Set Base (m-X)		Zmacs command	160
Set Fonts (m-X)		Zmacs command	160
Set Lowercase (m-X)		Zmacs command	160
Set Nofill (m-X)		Zmacs command	160
Set Package (m-X)		Zmacs command	156
Set Patch File (m-X)		Zmacs command	161
Set Tab Width (m-X)		Zmacs command	161
Set Variable (m-X)		Zmacs command	271
Set Visited File Name (m-X)		Zmacs command	139
Set Vsp (m-X)		Zmacs command	161
Show Buffer (m-X)		Zmacs command	135
Show Directory (m-X)		Zmacs command	149
Show File (m-X)		Zmacs command	150
Show File Properties (m-X)		Zmacs command	150
Show Keyboard Macro (m-X)		Zmacs command	258
Show Login Directory (m-X)		Zmacs command	149
Source Compare (m-X)		Zmacs command	142
Source Compare Merge (m-X)		Zmacs command	142
Trace (m-X)		Zmacs command	56
Update Attribute List (m-X)		Zmacs command	156
Variable Apropos		Zmacs command	269
Write File		Zmacs command	34
Example of Finding Out What a		Zmacs Command Does	51
Finding Out What a		Zmacs Command Does	51
		Zmacs Command: m- .	235
Finding Out About		Zmacs Commands	51
General Information-giving		Zmacs Commands	54
Introduction to		Zmacs Commands	6
Method for Searching for Appropriate		Zmacs Commands	53
More HELP Commands for Finding Out About		Zmacs Commands	53
Mouse-sensitive		Zmacs commands	67
Overview of Finding Out About		Zmacs Commands	51
Searching for Appropriate		Zmacs Commands	52
		Zmacs Commands for Finding Out About Flavors	281
		Zmacs Commands for Finding Out About Lisp	280
		Zmacs Commands for Finding Out About the State of	
		Buffers	278

- Zmacs Commands for Finding Out About the State of Zmacs 279
- Zmacs Commands for Formatting Text 35
- Zmacs Commands for Interacting with Lisp 282
- Zmacs Commands with HELP 51
- Zmacs Commands with Keyboard Macros 257
- Zmacs Commands with Keyboard Macros 257
- Zmacs Command Tables 7
- *zmacs-comtab*** 259
- Zmacs Echo Area 20
- Zmacs Edit Callers Commands 239
- Zmacs Edit Definition Commands 235
- Zmacs Editor Window 18
- Zmacs Electric PI1 Mode 177
- Zmacs Environment 251
- Zmacs Environment 252
- Zmacs Extended Commands 7
- Zmacs File Manipulation Commands 148
- Zmacs File Manipulation Commands 148
- Zmacs Format Commands 41
- Zmacs Fortran Mode 177
- Zmacs Fundamental Mode 176
- Zmacs Help 14
- Zmacs Help 14
- Zmacs Help Command Summary 277
- Zmacs Incremental Search 114
- Zmacs in Init Files 272
- Zmacs in Init Files 272
- Zmacs Key Bindings 267
- Zmacs Keyboard Macro 257
- Zmacs Keyboard Macros Work 257
- Zmacs Keystrokes 7
- Zmacs Lisp Mode 176
- Zmacs List Definition Commands 238
- Zmacs Macsyma Mode 177
- Zmacs Major Editing Modes 176
- Zmacs Major Mode 175
- Zmacs Major Modes 256
- Zmacs Major Modes 256
- Zmacs Major Modes 256
- Zmacs Manual 1
- Zmacs Manual 3
- Zmacs Manual 4
- Zmacs Manual 4
- Zmacs Manual 4
- Zmacs Manual Notation Conventions 9
- Zmacs Midas Mode 177
- Zmacs Minor Modes 253
- Zmacs Minor Modes 253
- Zmacs Minor Modes 254
- Zmacs Minor Modes Work 253
- Zmacs Mode Line 20
- Zmacs Notation Conventions 9
- Zmacs Notation Conventions 9
- Zmacs Notation Conventions 9
- Zmacs Notation Conventions and Examples 9
- Zmacs PI1 Mode 177
- Zmacs Region? 1C0
- Zmacs Reverse Incremental Search 115
- Zmacs Sentence Delimiters 73, 97
- Zmacs Sorting Commands 128

- Using the
 - Zmacs Speller 179
 - Zmacs Speller 180
 - Zmacs Speller Accept command 181
 - Zmacs Speller Accept Once command 181
- The
 - Zmacs Speller Menu 181
 - Zmacs Speller Prompt command 181
 - Zmacs String Search 116
 - Zmacs Teco Mode 177
 - Zmacs Text Mode 176
- Definition of a
 - Zmacs Variable 269
 - Zmacs variables 54
- Describing
 - Zmacs Variables 269
- Finding Out About
 - Zmacs Variables 269
- Listing
 - Zmacs Variables 269
- Settable
 - Zmacs Variables 270
- How to Specify
 - Zmacs Variable Settings 269
- Leaving
 - Zmacs Via the System Menu 44
 - Zmacs Window Commands 146
- Leaving
 - Zmacs with c-Z 45
- Entering
 - Zmacs with ed 12
- Entering
 - Zmacs with SELECT E 12
- Entering
 - Zmacs with the Mouse 12
- Leaving
 - Zmacs with the SELECT Key 44
- Entering
 - Zmacs with **zwei:edit-functions** 13
- Entering
 - Zmail 44
 - Zwei:*inhibit-fancy-loop indentation 216
 - zwei:add-words-to-spell-dictionary** function 194
 - zwei:command-store** 259
 - zwei:define-keyboard-macro** 259
 - zwei:defmajor** 256
 - zwei:delete-words-from-spell-dictionary** function 195
- Entering Zmacs with
 - zwei:edit-functions** 13
 - zwei:edit-functions** function 13
 - zwei:*file-versions-kept*** variable 153
 - zwei:*major-mode-translations*** variable 256
 - zwei:make-macro-command** 259
 - zwei:read-spell-dictionary** function 193
 - zwei:read-standard-spell-dictionaries** function 194
 - zwei:*send-mail-about-patch*** 247
 - zwei:*set-attribute-update-list*** global variable 159
 - zwei:*temp-file-type-list*** variable 153
 - zwei:*zmacs-comtab*** 259

[

[

[

c-X

- [Font Editor command 330
- [Zmacs command 28, 80

\

\

\

- @ \ Font Editor command 330
- @ \ text formatting command 39

]

]

]

c-X] Font Editor command 330
] Zmacs command 28, 80

^

^

^

c-X ^ Zmacs command 146
@ ^ text formatting command 39

