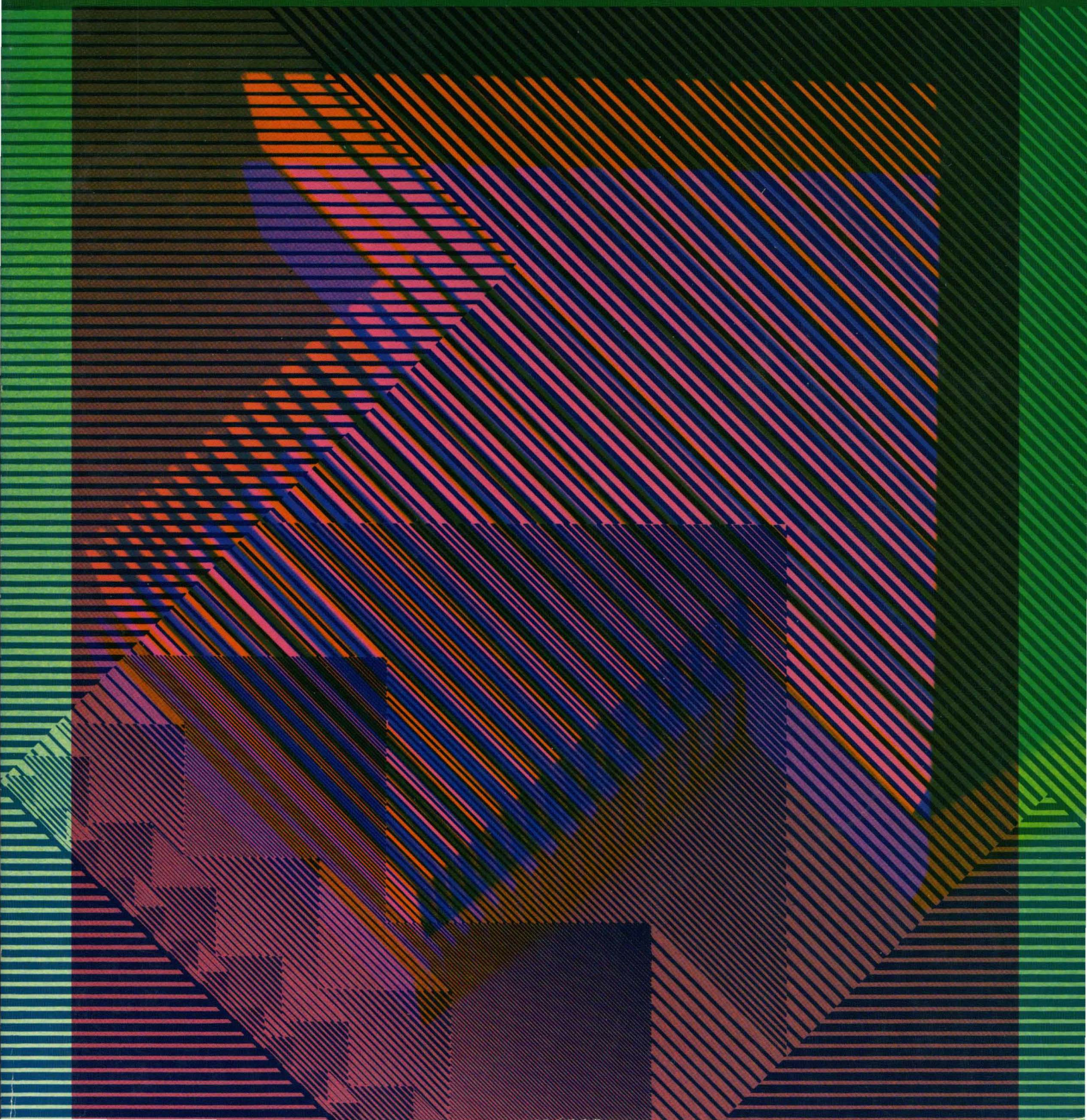
3 Text Editing and Processing

symbolics



3 Text Editing and Processing

symbolics

Cambridge, Massachusetts

Text Editing and Processing

996035

March 1985

This document corresponds to Release 6.0 and later releases.

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

Text typography: Century Schoolbook and Helvetica produced on Symbolics 3600-family computers from Bitstream, Inc., outlines; text masters printed on Symbolics LGP-1 Laser Graphics Printers.
Cover design: Schafer/LaCasse
Cover printer: W.E. Andrews Co., Inc.
Text printer: ZBR Publications, Inc.

Printed in the USA.

Printing year and number: 87 86 85 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 17
Inserting Text 22
Numeric Arguments 24
Introduction to Moving the Cursor 26
Introduction to Erasing Text 28
Creating and Saving Buffers and Files 30
Zmacs Commands for Formatting Text 33
Leaving Zmacs 40

3. Getting Help in Zmacs 41

Getting Out of Trouble 42
Finding Out About Zmacs Commands 44
The Editor Menu 49
More on the Minibuffer 51

4. Moving the Cursor in Zmacs 53

Overview of Moving the Cursor 54
Redisplaying the Window 55
Moving the Cursor with the Mouse 57
Motion Commands 60
Motion by Lisp Expression 65
Motion by Paragraph 68
Motion by Page 69
Motion with Respect to the Whole Buffer 70

5. Deleting and Transposing Text in Zmacs 71

Deleting Vs. Killing Text 72
Deleting and Transposing Characters 76

Deleting and Transposing Words 78
Deleting and Transposing Lisp Expressions 79
Deleting and Transposing Lines 81
Deleting Sentences 83

6. Working with Regions in Zmacs 85

What is a Zmacs Region? 86
Registers in Zmacs 89
Commands to Mark Regions 91
Region-manipulating Commands 94

7. Searching, Replacing, and Sorting in Zmacs 97

Searching in Zmacs 98
Locating and Replacing Strings Automatically 102
Tag Tables and Search Domains 106
Sorting 112

8. Manipulating Buffers and Files in Zmacs 113

Working with Buffers and Files 114
Selecting, Listing, and Examining Buffers 116
Buffer Commands 117
Appending, Prepending, and Inserting Text 124
Comparing Files and Buffers 125
Window Commands 129
File Manipulation Commands 131
Buffer and File Attributes 137
Dired Mode 144

9. Setting the Zmacs Major Mode 155

Major Editing Modes 156

10. Changing Case and Indentation in Zmacs 159

Changing Case 160
Indentation 162

11. Editing Lisp Programs in Zmacs 169

Introduction 170
Commenting Lisp Code 171
Evaluating and Compiling Lisp Programs 174
Parenthesizing Lisp Expressions 178
Expanding Lisp Expressions 179
Locating Source Code to Edit 180
Patching 186

12. Customizing the Zmacs Environment **193**

Overview 194
Built-in Customization Using Zmacs Minor Modes 195
Major Modes 198
Creating New Commands with Keyboard Macros 199
Key Bindings 208
How to Specify Zmacs Variable Settings 210
Customizing Zmacs in Init Files 213

Appendix A. Zmacs Help Command Summary **217**

II. Font Editor **219**

13. Font Basic Concepts **221**

13.1 Attributes of TV Fonts 221
13.2 Standard TV Fonts 223

14. Entering and Leaving FED **225**

15. Font Editor Basic Concepts **227**

15.1 FED, the Subsystem 227
15.2 Selecting a Font 230
15.2.1 Creating a New Font 230
15.2.2 Displaying Characters in the Font 233
15.3 Selecting a Character 233
15.3.1 From the Character Select Menu 233
15.3.2 By Creating a New Character 233
15.3.3 From the [Show Font] Display 233
15.3.4 With the c Command 233
15.3.5 By Renaming Characters 234

16. Drawing **235**

16.1 Drawing Characters with the Mouse 235
16.2 The Nonmouse Cursor 236

17. Viewing and Altering a Character in the Character Box **237**

17.1 What the Lines Mean 237
17.2 Altering the Character Box 238

18. The Gray Plane **239**

18.1 Getting Things Into Gray 239
18.1.1 With [Swap Gray] 239
18.1.2 With [Gray Char] 239
18.2 Merging Characters with the Gray Plane 240

19. Saving Characters and Pieces of Characters in Registers **241**

19.1 Saving a Drawing Into a Register 241
19.2 Retrieving the Contents of a Register 241
19.3 Retrieving the Black Plane While Manipulating Registers 241

20. Transformations **243**

20.1 Clearing the Drawing 243
20.2 Rotating Drawings 243
20.3 Reflecting Drawings 243
20.4 Moving the Drawing 246
20.5 Drawing Lines and Curves 246
20.6 Stretching and Contracting 246
20.6.1 Stretching a Drawing Horizontally 250
20.6.2 Contracting a Drawing Horizontally 250
20.6.3 Stretching a Drawing Vertically 250
20.6.4 Contracting a Drawing Vertically 250

21. The Sample String **253**

22. Adjusting the Display **255**

22.1 Positioning the Drawing 255
22.2 Setting the Box Size in the Drawing Pane 256
22.3 Setting the Height and Width of the Drawing Pane 256

23. Reading and Writing Files **257**

23.1 Reading Files 257
23.2 Writing Files 258

24. Command List **259**

24.1 Menu and Keyboard Commands 259
24.1.1 Configuration and Drawing Transformation 259
24.1.2 Gray Plane Menu Items 260
24.1.3 Outside World Interface Menu Items 260
24.1.4 Evaluating Forms From FED 261
24.2 Keyboard-only Commands 261
24.3 Mouse Sensitivities 262
24.3.1 The Drawing Pane 262
24.3.2 The Draw Mode Menu 262
24.3.3 The Sample Pane 262
24.3.4 The Character Select Pane 263
24.3.5 The Font Parameters Menu 263
24.3.6 The Register Pane 263
24.3.7 The List Fonts and Show Font Displays 263

III. Hardcopy System **265**

25. Printing and Hardcopy Commands **267**

25.1 Commands for Producing Hardcopy 267
25.1.1 Hardcopying From the System Menu 267
25.1.2 Hardcopying From Zmacs 267
25.1.3 Hardcopying From Zmail 268
25.1.4 Hardcopying From Dired 268
25.1.5 Hardcopying the Screen 268
25.1.6 Hardcopying From the File System Editor 268
25.2 Other Hardcopy Commands 269
25.2.1 Changing the Default Printer 269
25.2.2 Checking the Status of Hardcopy Devices 269

26. Customizing Hardcopy Facilities **271**

27. Hardcopy Functions **273**

Index **277**

List of Figures

Figure 1. Initial FED Display 226
Figure 2. Tall Configuration 231
Figure 3. Wide Configuration 232
Figure 4. [Rotate (R)] 244
Figure 5. [Rotate (M)] 245
Figure 6. [Reflect] 247
Figure 7. The X axis (-) 248
Figure 8. Moving the Drawing with [Move Black] 249
Figure 9. Stretching Horizontally 251
Figure 10. Contracting Horizontally 252

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 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.

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

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.

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

Introduction to Zmacs, *cont'd.*

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 in the Minibuffer".

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.

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,

Introduction to Zmacs, *cont'd.*

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.)

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.

Value	*Description*
t	The **ed** function enters the editor, creates an empty buffer, and selects it.
Pathname or string	The **ed** function enters the editor and

Entering Zmacs, *cont'd.*

	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 183.

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 41.

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 210. 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 whose names contain a string you specify. (The A refers to *apropos*, the name of the function that finds the commands and displays their 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 210.
HELP W	Displays the key binding for a command you specify. (The W refers to where.)
HELP SPACE	Repeats the last HELP command.

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.

Zmacs Help, *cont'd.*

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.

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

Introduction to Yanking

Yanking helps you to get back any text that you have typed or deleted, by retrieving it from a *history*. A history remembers commands and pieces of text, placing them in a *history list* in stack order, with the newer elements at the top of the history and the older elements toward the bottom.

Yanking commands yank back an element of a history from any position in the history list that you specify:

Yanking in the kill history:

c-0 c-Y	Displays the elements of the kill history (saved text). Click left on (*N* more elements in history.) to display those not shown.
c-Y	Yanks the first element in the kill history.
m-Y	After c-Y, yanks the previous element in the kill history. Subsequent m-Ys move down the kill history list.

Yanking in the command history:

c-0 c-m-Y	Displays the elements of the command history (editor commands that use the minibuffer in any

Zmacs Help, *cont'd.*

	way). Click left on (*N* `more elements in history.`) to display those not shown.
`c-m-Y`	Yanks the first element in the command history.
`m-Y`	After `c-m-Y`, yanks the previous element in the command history. Subsequent `m-Y`s move down the command history list.

For complete descriptions of killing and yanking: See the section "Deleting and Transposing Text in Zmacs", page 71.

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

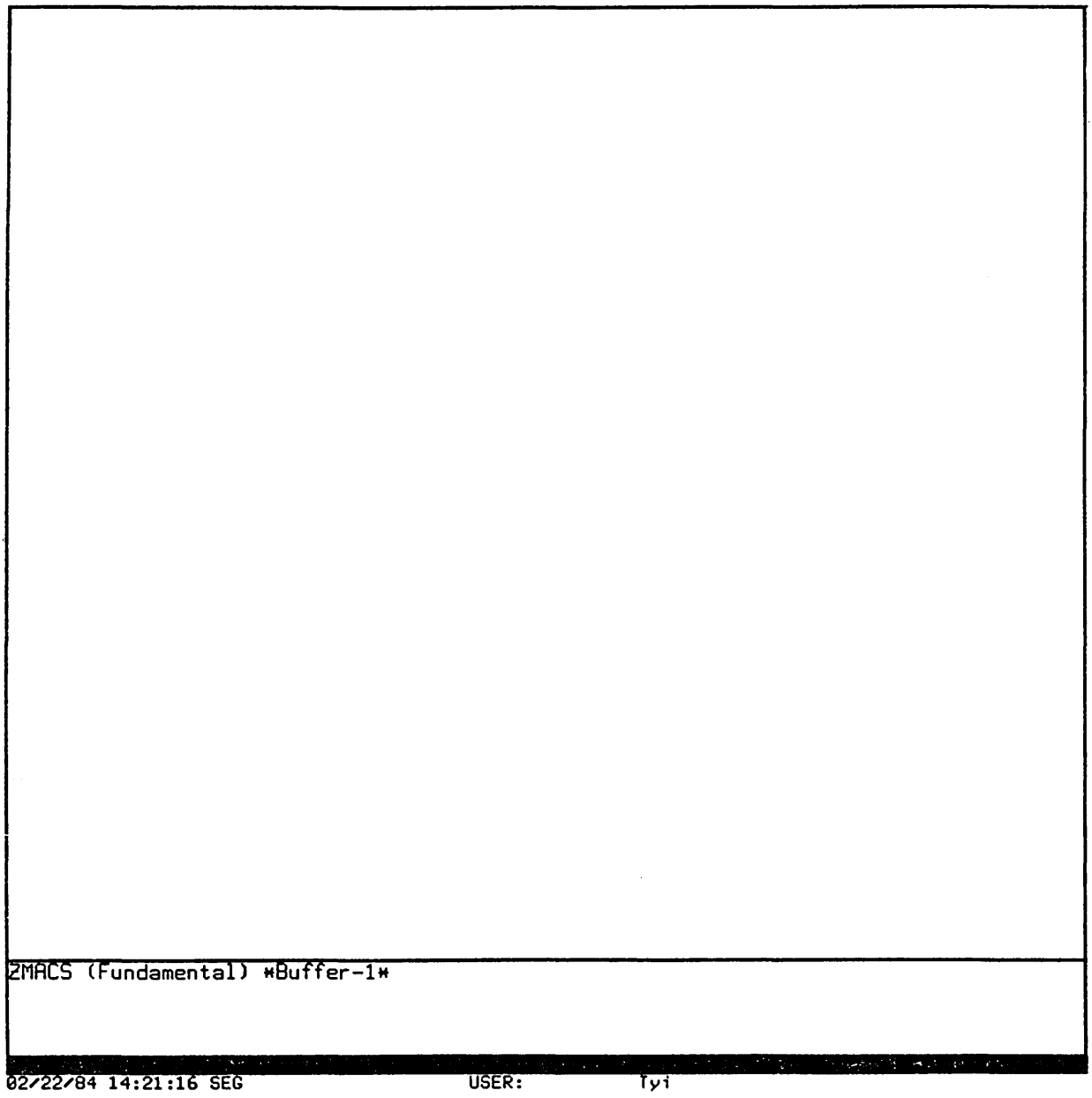
The small blinking rectangle, the *cursor*, usually appears somewhere within the buffer, showing the position of *point*, the location at which editing takes place. Although the cursor covers a single character, we consider point to be at the left edge of the cursor, between the character the cursor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



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:

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

Organization of the Screen, *cont'd.*

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 155.

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 195.

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.

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.

Organization of the Screen, *cont'd.*

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 113.

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 24.

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 72.

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 Zmacs 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 command Set Pop Mark, c-U takes different actions with the mark. (For a

Numeric Arguments, *cont'd.*

description of the Set Pop Mark command: See the section "Working with Regions in Zmacs", page 85.)

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 53.

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.*

c-V, SCROLL	Next Screen
Moves down to display the next screenful of text.	
m-V, m-SCROLL	Previous Screen
Moves up to display the previous screenful of text.	
m-<	Goto Beginning
Moves to the beginning of the buffer.	
m->	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 85.

Summary of Erasing Text

c-D — Delete Forward
Deletes the character after point.

RUBOUT — Rubout
Deletes the character before point.

m-D — Kill Word
Kills forward one word.

m-RUBOUT — Backward Kill Word
Kills backward one word.

m-K — Kill Sentence
Kills forward one sentence.

c-X RUBOUT — Backward Kill Sentence
Kills backward one sentence.

c-K — Kill Line
Kills to the end of the line or kills an end of line.

c-W — Kill Region
Kills region (from point to mark).

c-m-K — Kill Sexp
Kills forward over exactly one Lisp expression.

c-m-RUBOUT — Backward Kill Sexp
Kills backward over exactly one Lisp expression.

Introduction to Erasing Text, *cont'd.*

m-\ Delete Horizontal Space

Deletes any spaces or tabs around point.

c-X c-O Delete Blank Lines

Deletes any blank lines following the end of the current line.

m-^ 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 113.

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.

The other way to create another buffer is c-X c-F, Find File.

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

(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:

System	*Empty Buffer Name*
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, 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".

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.

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

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 Symbolics LGP-1 printer (no other printer is supported) by giving the Format command a numeric argument.

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 bulletted 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:

```
@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:

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

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.
center	Centers each line in an unfilled environment.
description	Outdents paragraphs with single spacing and wider margins in a filled environment.
display	Displays enclosed text in Roman (default) typeface and widens both margins in an unfilled environment.
enumerate	Moves the left margin to the right, displaying a number in the left margin for each paragraph.
equation	Displays equations in an unfilled environment

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

	using fixed-width typeface. It widens both margins.
example	Displays examples in an unfilled environment using fixed-width typeface. It widens both margins.
figure	Displays figures in an unfilled environment using Roman (default) typeface with no changes to the margins.
flushleft	Displays unfilled text aligned at the left margin. It recognizes and includes leading spaces.
flushright	Displays unfilled text aligned at the right margin. It ignores trailing spaces.
format	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, @\ and @>).
fullpagefigure	Displays figures in an unfilled environment using Roman (default) typeface with no changes to the margins.
fullpagetable	Displays tables in an unfilled environment using Roman (default) typeface with no changes to the margins.
g	Displays enclosed text in Greek typeface.
heading	Centers each line in boldface type in an unfilled environment.
i	Displays enclosed text in italics.
itemize	Moves the left margin to the right, displaying a bullet in the left margin for each paragraph.
majorheading	Centers each line in boldface type in an unfilled environment.
multiple	Keeps enclosed text together as a single item within itemize, enumerate, or description environments, regardless of intervening paragraph breaks.
outputexample	Displays typeout examples in an unfilled environment using fixed-width typeface. It widens the right margin.
p	Displays enclosed text in bold italics.
quotation	Displays enclosed text in a filled environment

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

	using Roman (default) typeface. It widens both margins.
r	Displays enclosed text in Roman typeface. For example to override the default typeface of the italicize environment: `@I(The Iliad @r[and] The Odyssey)` produces *The Iliad* and *The Odyssey*.
subheading	Displays each line in boldface type in an unfilled environment flush to the left margin.
table	Displays tables in an unfilled environment using Roman (default) typeface with no changes to the margins.
text	Displays enclosed text in a filled environment.
verbatim	Displays enclosed text in an unfilled environment in fixed-width typeface with no changes to the margins.
t	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.

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.

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

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()`.
@tabdivide	Sets tabs to divide text into the specified number of columns, for example, `@tabdivide(4)` 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.

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

@#	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 (@\, @>, or @=).
@>	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 (@\, @>, or @=).
@\	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.

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:

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

```
1.          *          *          *
2.    a          b          c          d
3.    e          f          g          h
4.LeftCenter                                    right
5.LeftCenter          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.

Format Region

Format Region (m-X)

Displays the contents of the region formatted as a text environment. With a numeric argument, it pops up a menu that asks whether you want to format the file on the screen or on the default printer.

Format Buffer

Format Buffer (m-X)

Displays the contents of the buffer, formatted as a text environment. With a numeric argument, it pops up a menu that asks whether you want to format the file on the screen or on the default printer.

Format File

Format File (m-X)

Displays the contents of the file, formatted as a text environment. With a numeric argument, it pops up a menu that asks whether you want to format the file on the screen or on the default printer.

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:

Key	*Program*
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 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.

c-G — Beep

Cancels the last command. It also cancels numeric arguments and region marking, except when given an argument. It cancels one

Getting Out of Trouble, *cont'd.*

thing at a time, so that if you've typed a number of commands or responses, you must use 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 71.

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 85.

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 HE!..P. 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 **com-forward**). (You don't need to know these internal names for basic editing.) The COM-xxx

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

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

When you can only guess at part of the name of a command by the action it performs, there is a command, HELP A, to help you scan all the available Zmacs commands to find the one you want.

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.

Method for Searching for Appropriate Zmacs Commands

To find the command you want, just press HELP A. 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 string that you enter contains a space, then Zmacs displays a short description of all the commands whose

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

names include 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 name is going to have the word *paragraph* in it.

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

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 210.

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

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

- 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.

---

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.

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

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.

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.

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 49.

Editor Menu Commands

The Editor Menu commands are:

Command	*Description*
Arglist	Prints the argument list of the specified function: See the section "General Information-giving Zmacs Commands", page 46.
Edit Definition	Prepares to edit the definition of a specified function: See the section "Editing Lisp Programs in Zmacs", page 169.
List Callers	Lists all functions that call the specified function: See the section "Editing Lisp Programs in Zmacs", page 169.
List Definitions	Displays the definitions in a specified buffer: See the section "Editing Lisp Programs in Zmacs", page 169.
List Buffers	Prints a list of all the buffers and their associated files: See the section "Manipulating Buffers and Files in Zmacs", page 113.
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 113.
Split Screen	Makes several windows split among the buffers as specified: See the section "Manipulating Buffers and Files in Zmacs", page 113.

The Editor Menu, *cont'd.*

Compile Region	Compiles the region, or if no region is defined, the current definition: See the section "Editing Lisp Programs in Zmacs", page 169.
Indent Region	Indents each line in the region: See the section "Changing Case and Indentation in Zmacs", page 159.
Change Default Font	Sets the default font: See the section "Working with Regions in Zmacs", page 85.
Change Font Region	Changes the font for the region: See the section "Working with Regions in Zmacs", page 85.
Uppercase Region	Changes any lowercase characters in the region to uppercase: See the section "Working with Regions in Zmacs", page 85.
Lowercase Region	Changes any uppercase characters in the region to lowercase: See the section "Working with Regions in Zmacs", page 85.
Indent Rigidly	Shifts text in the region sideways as a unit: See the section "Changing Case and Indentation in Zmacs", page 159.
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 165.

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-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.

After c-m-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 74.

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, or c-m-Y). The retrieved text that was yanked by that command is replaced by the previous element of the relevant 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.

More on the Minibuffer, *cont'd.*

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

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 85.) 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 redisplays 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 redisplays 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-V, 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 210.)

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:

- Appears just above the bottom line of the screen
- Normally stands out in reverse video
- Contains documentation on the current meaning of mouse clicks

In a regular Zmacs buffer, the mouse documentation line offers the following options:

Notation	*Description*
L:Move point	Performs two separate actions: • Relocates the cursor: position the mouse cursor to the desired location and click left. • Makes a region: position mouse cursor to desired location, click left (keeping the button down), move mouse cursor to end of region and lift the button up.
L2:Move to point	Relocates the mouse cursor near the cursor: click left twice.
M:Mark thing	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, as delimited by nonalphanumeric characters. 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

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

printname, depending on mode. (For a complete description of *marking* regions: See the section "Working with Regions in Zmacs", page 85.

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 to be 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 put before the inserted object, if needed.

M2: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 85.
R:Menu	Displays a Zmacs menu offering mouse-sensitive Zmacs commands.
R2:System Menu	Displays a System menu.

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

M-F moves the cursor forward one word, whereas M-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

M- -13F moves point backward 13 words. M-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 155.)

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.

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 Commands, *cont'd.*

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-A Backward Sentence

Moves the cursor backward one sentence.

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

M-A always places the cursor at the beginning of a sentence. If the cursor is in the middle of a sentence, M-A 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

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

Motion Commands, *cont'd.*

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

Motion by Lisp Expression, *cont'd.*

c-m-A
c-m-[Beginning Of Definition

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.

c-m-E
c-m-] End Of Definition

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.

m-) 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 94.

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 white space only do not save it.

What Histories Save

Zmacs uses several histories:

Type	*Description*
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 (m-X) and related commands. See the section "Searching, Replacing, and Sorting in Zmacs", page 97.
Buffer	History of editor buffers visited in this window. See the section "Manipulating Buffers and Files in Zmacs", page 113.
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, m-RUBOUT, are not recorded in the history.
Definition	History of names of definitions that have been typed.

History lengths are limitless 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

c-0 c-Y

Displays the elements of the kill history (saved text) in a typeout window:

```
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.)
```

Viewing the Editor Command History

c-0 c-m-Y

Displays the elements of the editor command history (commands typed) in a typeout window:

```
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.

Deleting Vs. Killing Text *cont'd.*

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 it.

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-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.

- After c-m-Y, m-Y replaces what was yanked with a previous element of the same history, in this case, another minibuffer command.
- After c-Y, m-Y replaces what was yanked with a previous element of the same history, in this case, the previous saved text.

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, or c-m-Y). The retrieved text that was yanked by that command is replaced by the previous element of the relevant 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 list. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

Deleting Vs. Killing Text *cont'd.*

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

Deleting and Transposing Characters, *cont'd.*

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 61.

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 left of point, moves forward *n* expressions, and reinserts the deleted expression. With a negative numeric argument, it exchanges

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

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-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.

Deleting the Current Line

c-K Kill Line

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

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.

Deleting and Transposing Lines, *cont'd.*

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 88.

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 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 91.

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

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

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

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-Us are typed in front of it:

Argument	*Action Taken*
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-Us	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.

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.

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.
```

View 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 88. However, several convenient commands mark different specific amounts of text:

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

Marking Words

m-@ 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

c-m-@ 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 65.

c-m-H 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.

Commands to Mark Regions, *cont'd.*

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.

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

m-W Save Region

Puts region on kill history list without deleting it. For information on kill merging and the Append Next Kill command, c-m-W: See the section "Kill Merging", page 75.

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 74.

Compiling a Region

c-sh-C Compile Region
Compile Region (m-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 (m-X)

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

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 m-Q, m-G, or Auto Fill Mode formatting. In addition, the *fill prefix*, if set, is inserted:

- 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

Region-manipulating Commands, *cont'd.*

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.

---

m-Q Fill Paragraph

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

---

m-G Fill Region

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

---

c-X . 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)

Lowercase Code in Region (m-X)

See the section "Changing Case of Regions in Zmacs", page 160.

---

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-Search:. When it locates the string, it puts the cursor after it so that repeated c-Ss locate subsequent occurrences of the default string in the buffer.

Searching in Zmacs, *cont'd.*

RUBOUT	Removes a character and backs up the search to the last match.
ESCAPE	When typed before any search characters, switches to String Search. See the section "Zmacs String Search", page 100.
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	Repeats the search.
c-R	Reverses the search to search backwards.

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).

Reverse Incremental Search

c-R, Reverse Incremental Search, works exactly the same way as c-S, except that it searches *backward* towards the top of the buffer from point, instead of forward.

c-R Reverse Incremental Search

Searches for a character string while you type it, searching backward to the beginning of the buffer. It prompts for a string in the echo area with `Reverse I-Search:`. As you type characters in, c-R displays the accumulating string in the echo area and searches for it at the same time. If no string is found, it displays `Failing Reverse I-Search:`. When it locates the string, it puts the cursor in front of it so that repeated c-Rs locate previous occurrences of the default string in the buffer.

RUBOUT	Removes a character and backs up the search to the last match.
ESCAPE	When typed before any search characters, switches to Reverse String Search. See the section "Zmacs String Search", page 100.
END or ESCAPE	Exits the search.
c-G	Exits the search and returns to original starting point in the buffer.

Searching in Zmacs, *cont'd.*

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.
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.

Searching in Zmacs, *cont'd.*

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-Z, Replace String, searches forward for a string and replaces that string with another. c-Z 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-Z, Query Replace. m-Z 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-Z locates each occurrence and lets you decide what to do about each one.

Making Global Replacements in Zmacs

c-Z Replace String

Replace String (m-X)

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 210.)

Querying While Making Global Replacements in Zmacs

m-Z Query Replace

Query Replace (m-X)

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

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

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 210.)

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 time, where the strings can be characters, words, or phrases. (See the section "Querying While Making Global Replacements in Zmacs", page 102.) 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.

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

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 102.) 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 102.)

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 106. Then you can perform an operation. See the section "Performing Operations with Tag Tables", page 107. 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 107. 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 called `*Tag-Table-`*N*`*`, which contains a list of the names of all the buffers. See the section "Support Buffers", page 109.

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

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.

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 110.) 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 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:

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

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 210.)

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 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-X)

Replaces occurrences of any number of strings with other strings within the tag table files, asking about each change. The current

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

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-X)

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-X)

Creates a tag table by reading in a tag file. 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 109. Each contains the names of the files.

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

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

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.

For a description of the Edit Definition and Edit Callers commands: See the section "Editing Lisp Programs in Zmacs", page 169.

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

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

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 (m-X)

Sorts the region alphabetically by lines.

Sort Paragraphs (m-X)

Sorts the region alphabetically by paragraphs.

Sort Via Keyboard Macros (m-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 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

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

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-m-L — Select Previous Buffer

Selects a previously selected buffer. With a numeric argument *n*, it selects the *nth* 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-m-L — Select Default Previous Buffer

With a numeric argument *n*, this is exactly the same as c-m-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 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

Buffer Commands, *cont'd.*

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 all 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

Edit Buffers (c-m-X) 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 213.) It contains one line for each of the buffers in the editor.

Edit Buffers (c-m-X)

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 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:

RUBOUT Undeletes buffer above the cursor.

Buffer Commands, *cont'd.*

SPACE	Selects the specified buffer immediately.
D	Marks the buffer for deletion (K, c-D, c-K 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 m-X).

Viewing a Buffer

View Buffer is for when you want to just look at a buffer, not edit it.

c-X V View Buffer

View Buffer (m-X)

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 a screenful, it pauses between screenfuls, displaying a --MORE-- message at the bottom.

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

Anything else exits and is executed as a command.

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.

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.

Buffer Commands, *cont'd.*

Saving Buffers

Save All Files (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 old.lisp /dass/pubs/pgs VIXEN:? (Y or N).`

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".

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.

Buffer Commands, *cont'd.*

c-X c-V Visit File

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

Buffer Commands, *cont'd.*

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

Buffer Commands, *cont'd.*

Destroying Buffers

c-X K — Kill Buffer

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

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-X)

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 A — 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:

Option	*Action*
1	Leaves the first alternative in the text, redisplays the contents, and asks for confirmation of change.

Comparing Files and Buffers, *cont'd.*

2	Leaves the second alternative in the text, redisplays the contents, and asks for confirmation of change.
*	Leaves both alternatives in the text, redisplays 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:

Option	*Action*
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.

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).

Comparing Files and Buffers, *cont'd.*

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)

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-X)

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

Comparing Files and Buffers, *cont'd.*

Source Compare Merge Installed Definition (M-X)

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 O — 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-V — Scroll Other Window

Scrolls the other window up several lines. By default, it scrolls the same way as c-V. With no argument, it scrolls a full screen. With just a minus sign as an argument (c-m- -V), 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

Host	*Directory string*	*Result*
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 (*bin*) 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/View Directory

Show Directory (m-X)

View Directory (m-X)

Prompts for the name of a directory and and displays the directory contents for viewing only in the typeout window. If there is more than a screenful, it pauses between screenfuls 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.

Viewing a File

Show File/View File is for when you just want to look at a file, not edit it.

Show File (m-X)

View File (m-X)

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

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

File Manipulation Commands, *cont'd.*

Anything else exits and is executed as a command.

Viewing the Properties of a File

View File Properties (m-X)

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-X)

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:

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

Hardcopying a File

Hardcopy File (m-X)

Sends a file to the local hardcopy device for printing.

Renaming a File

Rename File (m-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.

File Manipulation Commands, *cont'd.*

Copying a File Into Another

Copy File (m-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 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.

File Manipulation Commands, *cont'd.*

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

           SCRC:<LMFS>FILES.LISP.147 is copied into
           ff:>sys-hold>scrc-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.

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:

File Manipulation Commands, *cont'd.*

- 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 210.)

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 (Zetalisp) 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.

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

Buffer and File Attributes, *cont'd.*

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 210.) You can set the variable in your init file by using the internal form of its name. (See the section "Creating an Init File", page 213.)

For example, in your init file:

Buffer and File Attributes, *cont'd.*

```
(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:

Location	*"Y"*	*"N"*
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 **base** and **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 is assumed to be Zetalisp.
- If there is a Syntax: Common-Lisp attribute, 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 Zetalisp. 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.

Buffer and File Attributes, *cont'd.*

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 *Reference Guide to Symbolics-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 **readtable**, which is known as ***readtable*** in Common Lisp.
Package	**user** 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 **user** rather than **cl-user**.

The following example shows the attributes that should be in an SCL file's attribute line:

```
;;; -*- Mode:Lisp; Syntax:Common-Lisp; Package:CL-USER -*-
```

Set 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

Buffer and File Attributes, *cont'd.*

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.

Value	*Meaning*
:ask	Always asks whether to update the attribute list.
nil	Never updates the attribute list.
t	Always updates the attribute list.

Set *attribute* (M-X)

where *attribute* is one of the following: Backspace, Base, Default File Name, Fonts, Key, Lowercase, Nofill, Package, Patch File, Syntax, Tab Width, Variable, Visited File Name, 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.

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 **ibase** that the Lisp reader uses when reading forms from the file. Thus, Base controls the **ibase** used when you evaluate or compile parts of the buffer, *and* controls the value of **base** for printing during evaluating all or part of the buffer. This value does not affect the values of either **base** or **ibase** in the Lisp Listener you get by using SUSPEND.

Buffer and File Attributes, *cont'd.*

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 213.) 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	The Set Nofill command has a default value of **nil**, which means that whatever autofilling behavior you specify prevails. When Nofill is anything else (you answer `Y` to its query), it means that autofilling is not appropriate for people who specify the mode of "autofilling if appropriate".
	
	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.
	
	To have autofilling turned on by default, use the following in your init file (See the section "Creating an Init File", page 213.):
	
	`(login-forms`
	`  (setf zwei:text-mode-hook`
	`   'zwei:auto-fill-if-appropriate))`
	
	People who do not want it never get it by default.

Patch-File	The Set Patch File command has a default value of **nil**, which means that the file does not contain patches. When a file is classified as containing patches (you answer `Y` to its query), **fdefine** 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 **defvar** (which becomes **setf**).
Tab-Width	The Set Tab Width command (default 8 characters) specifies how many spaces the editor uses between "tab stops".
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
- $ if the file has been marked against reaping
- @ if the file has been marked against deletion
- The file's creation date

Dired Mode, *cont'd.*

- 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 121.) 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 147.)

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:

Character	*Action*
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 -*- 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).
A	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.
*n*H	Marks excess versions of the file for deletion (argument means whole directory).
L	Loads the file into Lisp.
*n*N	Moves to the next file with more than *n* versions (see the Zmacs variable File Versions Kept). (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings", page 210.)
P	Prints the file on the standard hardcopy device.
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:

Dired Mode, *cont'd.*

: 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.

Q or X — Abort out of Dired.

E — Delete files and expunge directory. This is meaningful for file systems in which there is undeletion, such as TOPS-20, TENEX, and the Lisp Machine file system. This command is useful if you use Dired to free up disk space, since the disk space is not deallocated until the directory is expunged.

Dired Mode, *cont'd.*

Dired Exit performs those actions and returns to the previous buffer.

ABORT — Dired Abort

Leaves Dired mode at once, without performing any actions on marked files. You can also just switch to another buffer.

Online Documentation for Dired

If you do not have a manual and cannot remember what the commands do, just press HELP.

? — Dired Help
HELP

Displays a short table explaining the Dired commands.

Dired Menu

Click right in Dired to display the Dired menu, which offers to perform the following actions on the listing:

```
Sort by reference date (up)
Sort by reference date (down)
Sort by creation date (up)
Sort by creation date (down)
Sort by file name (up)
Sort by file name (down)
Sort by file size (up)
Sort by file size (down)
Dired Automatic
Dired Automatic All
Dired Change File Properties
Dired Describe Attribute List
```

See the section "Deleting Multiple File Versions in Dired", page 151. See the section "Changing File Properties in Dired", page 149. See the section "Viewing File Attributes in Dired", page 149.

Loading a File in Dired

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.

Dired Mode, *cont'd.*

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
- 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.

Dired Mode, *cont'd.*

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 and move backward with BACKSPACE.

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.

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.

Dired Mode, *cont'd.*

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), reprieves 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 reprieves the file on the immediately preceding line, positioning point on that line.

With a numeric argument of *n*, it reprieves the files on the next *n* lines including the current line.

RUBOUT Dired Reverse Undelete

Reprieves the file on the preceding line.

With a numeric argument of *n*, it reprieves 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:

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 210.)

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.

Dired Mode, *cont'd.*

Setting Generation Retention Count

C 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

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 Mode, *cont'd.*

!	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 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 137.)
- By having Zmacs do it for you when you read a file with c-X c-F. 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".

Text Mode

Sets things up for editing English text. It puts Tab-To-Tab-Stop on TAB.

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.*

Major Editing Modes, *cont'd.*

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. 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 (m-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*+1st 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 loop Macros

The loop *Indentor*

Zwei now indents code within a **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 **loop** keyword clauses. This new indentation style is a major incompatible change in the 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.

Indentation, *cont'd.*

To turn off the new **loop** indentor, including 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 indentor.

How to Use the **loop** *Indentor*

Use the **loop** indentor the same way as always: type a token on a line of code inside a **loop** and then press TAB, which correctly indents the code.

The usual sequence:

```
LINE finally TAB
```

(substitute any other **loop** word for `finally`) reindents based on the new knowledge that this is a `finally` line rather than a body line. The **loop** indentor always ignores comments.

Loop Indentor 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 **loop** indentor 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))
```

Loop Indentor Example 2

The **loop** indentor second guesses on a few things, but gets them right after you type a token on a line and press TAB. For example:

Indentation, *cont'd.*

```
(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))
```

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.

Indentation, *cont'd.*

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.

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-m-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

Indentation, *cont'd.*

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 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-M-O 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.

Indentation, *cont'd.*

Inserting Blank Line

`c-O` Make Room

Inserts a blank line after point. With a numeric argument *n*, it inserts *n* blank lines.

Deleting Blank Line

`c-X c-O` Delete Blank Lines

Deletes any blank lines around the end of the current line.

11. 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 "Programming Development Tools and Techniques". 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".

Commenting Lisp Code

Overview

Zmacs differentiates between the different comment indicators for different major modes. Comments in Lisp begin with a semicolon. The Lisp reader ignores everything between a (significant) semicolon and the next newline. By convention, there are three kinds of comments, beginning with one, two, and three semicolons:

- Comments beginning with a single semicolon are placed to the right of a line of code, start in a preset column (the *comment column*), and describe what is going on in that line.
- A comment with two semicolons is a long comment about code within a Lisp expression and has the same indentation as the code to which it refers. It describes the function of a group of lines.
- A comment headed by three semicolons is normally placed against the left margin, and describes a large piece of code, such as a function or group of functions.

This section outlines Lisp commenting conventions and explains Zmacs commands for manipulating comments.

Indenting for Comment

c-; Indent For Comment
m-;

If the current line has no comment, moves point out to the comment column (inserting spaces to get there, if necessary) and starts a comment by inserting a semicolon there. If the current line already has a comment, it indents it correctly and leaves point at the beginning of it. Zmacs positions the various kinds of comments appropriately. If a comment begins at the left margin, it leaves it there.

With a numeric argument *n*, it realigns any comments on the next *n* lines, including the current line, but does not create any new comments.

If a comment cannot be positioned at the comment column because the associated line of code is too long, comments are moved to the right until they are clearly separated from the code.

Killing a Comment

c-m-; Kill Comment

If the current line has a comment, deletes it.

Commenting Lisp Code, *cont'd.*

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 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.

Commenting Lisp Code, *cont'd.*

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.

A numeric argument activates lines in the region that have been commented out. When any part of the line is part of the region, it removes commenting from around that line. This assumes that any comment starting in column 1 is fair game. It stops when it encounters a line that does not begin the way a comment would, even if more lines that have been commented out remain in the region. It does keep the remainder of the region in this case, so that you can resume.

Uncomment Region (m-X)

Removes all comments from lines whose beginnings are contained in the region.

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.

c-m-sh-E — Evaluate Region Verbose

Evaluates the region. When no region has been defined, it

Evaluating and Compiling Lisp Programs, *cont'd.*

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.)

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).

Evaluating and Compiling Lisp Programs, *cont'd.*

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 compiles 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.

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

Evaluating and Compiling Lisp Programs, *cont'd.*

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 66.

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.

M-. Edit Definition

Edits the newest 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 newest 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 M-. to edit another definition of the same object. You can repeat this until there are no more definitions of that object.

The prompt has three helpful features: selection by mouse, context default, and completion (for definitions already in the buffer). You can specify a definition by typing the name into the minibuffer or clicking left on a name already in the buffer. If you just press END instead of typing a function name, Zmacs assumes that the function you want is the one at the front of the innermost expression containing point. This default is displayed with the prompt.

Locating Source Code to Edit, *cont'd.*

Zmacs finds definitions this way:

- If the definition is in the current buffer, it moves point there.
- If the definition is in a different buffer, it changes buffers to get to the definition and moves point there.
- If the definition is in a file that has not been read into a Zmacs buffer, Zmacs goes out to the file system to get it, creating a new buffer and reading in the file, and then moves point to the definition.

When a symbol has more than one definition (for example, **list** might be defined both as a function and as a global variable), Zmacs finds all the definitions, but only presents the first one for editing. Zmacs remembers the other definitions, and tells you about them with a message in the echo area. When you have finished with the first definition, you can look at the next by invoking m-. with a numeric argument. Each time you do this, you bring up a new definition to be edited, until you run out of definitions. m-. displays `No more definitions` if you try to continue.

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 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.

Locating Source Code to Edit, *cont'd.*

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 110.

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 110.

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.

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.

Locating Source Code to Edit, *cont'd.*

List Changed Definitions (m-X)

Displays a list of any definitions that have been edited in any buffer. Use c-. (Next Possibility) to start editing the definitions in the list. See the section "Displaying the Next Possibility", page 110.

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 110.

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.

Edit Callers (m-X)

Prepares for editing all functions that call the specified one. The

Locating Source Code to Edit, *cont'd.*

prompt is the same kind that Edit Definition gives you. It reads a function name via the mouse or from the minibuffer with completion. By default, it searches the current package. You can control the package being searched by giving the function an argument. With an argument of c-U, it searches all packages; with c-U c-U, it prompts for the name of a package to search. It selects the first caller; use c-. (Next Possibility) to move to a subsequent definition. See the section "Displaying the Next Possibility", page 110.

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.

By default, it searches the current package. You can control the package being searched by giving the function an argument. With an argument of c-U, it searches all packages. With two c-Us, it prompts for the name of a package.

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.

With an argument of c-U, it lists all the callers in every package. With two c-Us, it prompts for the name of a package to search.

Multiple List Callers (m-X)

Lists all the functions that call 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 110.

Locating Source Code to Edit, *cont'd.*

By default, it searches the current package. You can control the package being searched by giving the function an argument. With an argument of c-U, it searches all packages. With two c-Us, it prompts for the name of a package.

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. View 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 **make-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 (m-X).
4. Add the changed code to the patch buffer by using Add Patch (m-X), Add Patch Changed Definitions of Buffer (m-X), or Add Patch Changed Definitions (m-X).
5. Finish the patch, using Finish Patch (m-X), or abort the patch, using Abort Patch (m-X).

Commands provided for initiating a patch are Start Patch (m-X), Start Private Patch (m-X), and Add Patch (m-X).

Start Patch (m-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 (m-X), Add Patch Changed Definitions (m-X), or Add Patch (m-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.
- Use the existing buffer and do not start a new patch.

Patching, *cont'd.*

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. These commands add only to the current patch buffer and warn you if you try to add code from one system to a patch for another.

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:

Response	*Action*
Y	Patches the definition.

Patching, *cont'd.*

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:

Response	*Action*
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.

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 View Patches commands to keep track of patches.

Patching, *cont'd.*

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.

View Patches (m-X)

View 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.

View 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.

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 **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.

Patching, *cont'd.*

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.

In 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.

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.

Patching, *cont'd.*

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 other way except by 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 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.

12. 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
- 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 208.

For information about setting minor modes permanently: See the section "Setting Mode Hooks in Init Files", page 214.

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.

Because of the way Auto Fill Mode works, you will often find

Built-in Customization Using Zmacs Minor Modes, *cont'd.*

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.

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.

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 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.

Built-in Customization Using Zmacs Minor Modes, *cont'd.*

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:defmajor** 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.

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 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.

Creating New Commands with Keyboard Macros, *cont'd.*

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.

Viewing a Keyboard Macro

To see the keyboard macro, use View Kbd Macro (m-X), which prints the macro at the top of your screen.

View Kbd 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.

```
c-X (
c-A
;
c-N
c-4 c-X )
```

Creating New Commands with Keyboard Macros, *cont'd.*

For information about setting key bindings permanently: See the section "Zmacs Key Bindings", page 208.

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.

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:

Creating New Commands with Keyboard Macros, *cont'd.*

```
(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.

`c-X Q` Kbd Macro Query

Allows user interaction on each iteration of macro, similar to Query Replace (`m-X`). While defining a keyboard macro, press `c-X Q` 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 `c-X (` and `c-X )` 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.
c-R	Enters editing mode; `c-m-FUNCTION R` resumes the keyboard macro.

Creating New Commands with Keyboard Macros, *cont'd.*

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 112.

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.

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

Creating New Commands with Keyboard Macros, *cont'd.*

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

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 208.

Creating New Commands with Keyboard Macros, *cont'd.*

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 200.)
R	Ends a macro definition (same as c-X) — See the section "Ending a Keyboard Macro", page 200.)
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 202.)
A	Steps though 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

Creating New Commands with Keyboard Macros, *cont'd.*

c-m-FUNCTION U allows typein while defining a macro that does not get stored in the macro, hence does not 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.

```
  you type:  c-X (
Minibuffer:  Macro-level: 1 *
  you type:  c-m-FUNCTION A
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-m-FUNCTION A, and this time, c-m-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 A
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-X. 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-X)

Installs a specified command on a specified key. 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.

Key Bindings, *cont'd.*

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 (m-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**.

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.

How to Specify Zmacs Variable Settings, *cont'd.*

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 *lispm-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 **setf** macro. For a description of symbol macros: See the section "Symbol Macros" in *Reference Guide to Symbolics-Lisp*. For a description of the **setf** macro: See the macro **setf** in *Reference Guide to Symbolics-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.

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.

Customizing Zmacs in Init Files, *cont'd.*

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 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.

Customizing Zmacs in Init Files, *cont'd.*

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.

Balanced Quotation Marks and Asterisks

```
ZWEI:(SET-COMTAB *STANDARD-COMTAB*
                 '(#\m-/" COM-MAKE-/(/)
                   #\c-m-/" COM-MOVE-OVER/)
                   #\m-/* COM-MAKE-/(/)
                   #\c-m-/* COM-MOVE-OVER-/)
                   ))
```

This defines commands to insert balanced pairs of quotation marks or asterisks into the buffer. For example, you can type an asterisked special variable name as `m-* FOO`, which inserts `*FOO*` into the buffer, ensuring that one does not forget to type the trailing asterisk.

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 (HELP A, m-X)
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

Brief Documentation (c-sh-D)
Describe Variable At Point (c-sh-V)
Edit Callers (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 (m-sh-D)
Multiple Edit Callers (m-X)
Multiple List Callers (m-X)
Quick Arglist (c-sh-A)
Where Is Symbol (m-X)

Zmacs Commands for Finding Out About Flavors
Describe Flavor (m-X)
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-sh-C, m-X)
Compile File (m-X)
Compile Region (c-sh-C, m-X)
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-sh-E, m-X)
Evaluate Into Buffer (m-X)
Evaluate Minibuffer (ESCAPE)
Evaluate Region (c-sh-E, m-X)
Evaluate Region Hack (m-X)
Evaluate Region Verbose (c-m-sh-E)
Load Compiler Warnings (m-X)
Macro Expand Expression (c-sh-M, m-X)
Trace (m-X)
Quit (c-Z)

PART II.

Font Editor

13. 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.

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**" in *Program Development Utilities*.

The **tv** package contains the window system, which includes fonts for screen display (as opposed to fonts for hardcopying).

13.1 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.

Char-exists Table Font Attribute

The *char-exists* table is an **art-1b** array for each font. It has a **1** for each character that actually exists in the font, and a **0** for other characters. 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.

13.2 Standard TV Fonts

You can use `Show Font HELP` in the Lisp Listener or the List Fonts (m-X) command in Zmacs to get a list of all the fonts that are currently loaded into the Lisp environment. The **fonts** package contains the names of all fonts. Here is a list of some of the useful fonts:

fonts:cptfont This is the default font, used for almost everything.

fonts:jess14 This is the default font in menus. It is a variable-width rounded font, slightly larger and more attractive than medfnt.

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 **cptfont**.
fonts:medfnb	This is a bold version of **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 *"More above"* and *"More below"*.
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 **hl10**, used for selected items in Choose Variable Values windows.

14. 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 **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 **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).

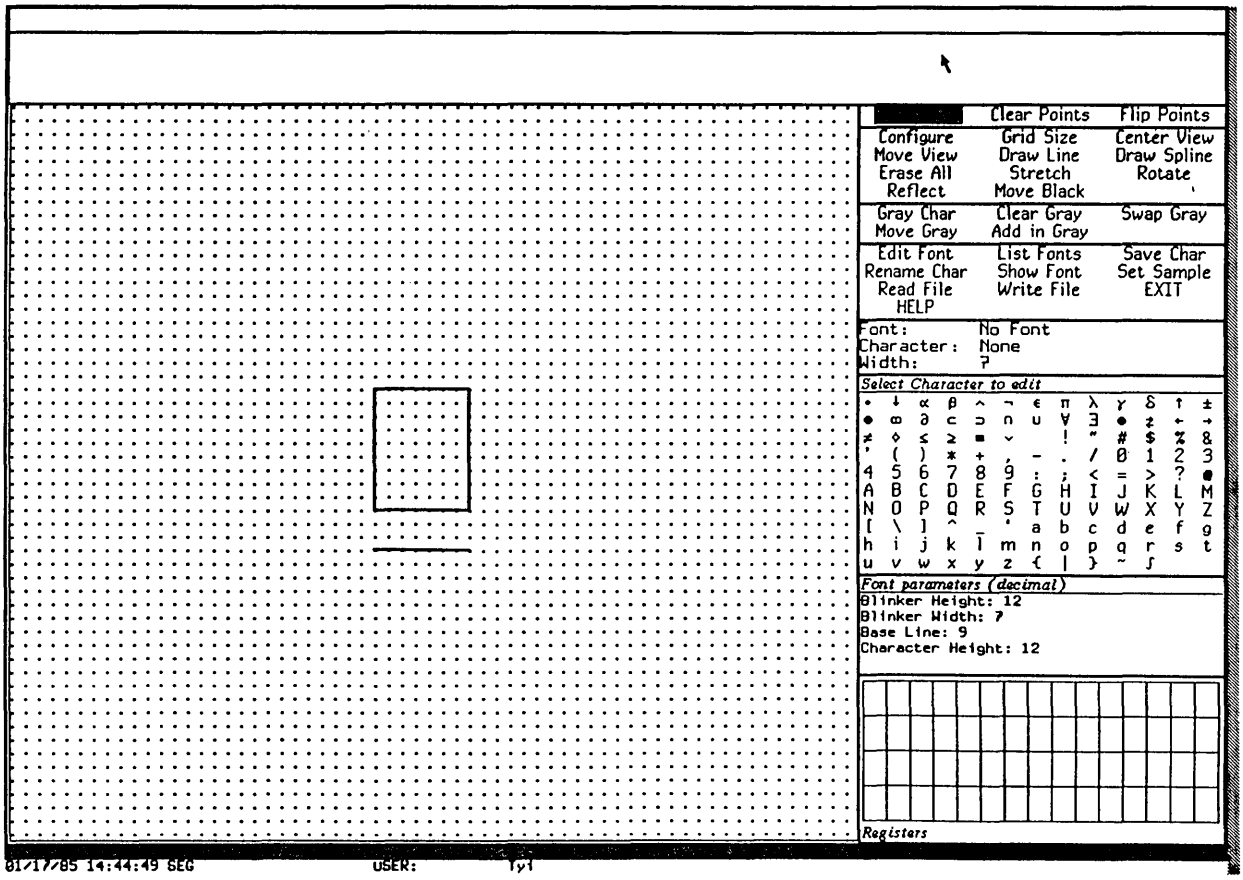


Figure 1. Initial FED Display

15. Font Editor Basic Concepts

15.1 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 235.

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 237.

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 235.

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 259. 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 239.

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 **greek9** with [Show Font], you see that the omega character in **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 **base** and **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 "FED Registers".

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.

15.2 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 257.

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 225.
- You can select the [Edit Font] menu item, which prompts for the font name in 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.

15.2.1 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.

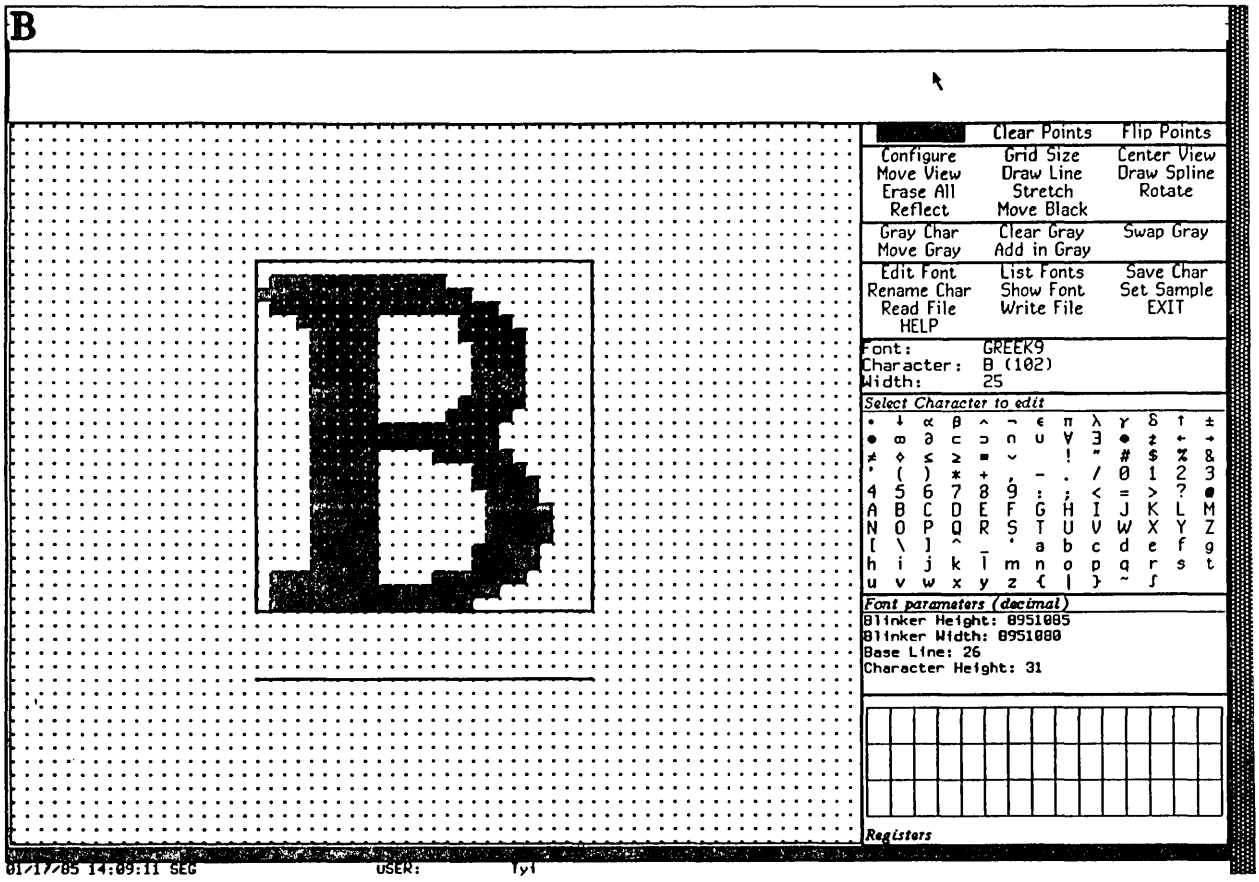


Figure 2. Tall Configuration

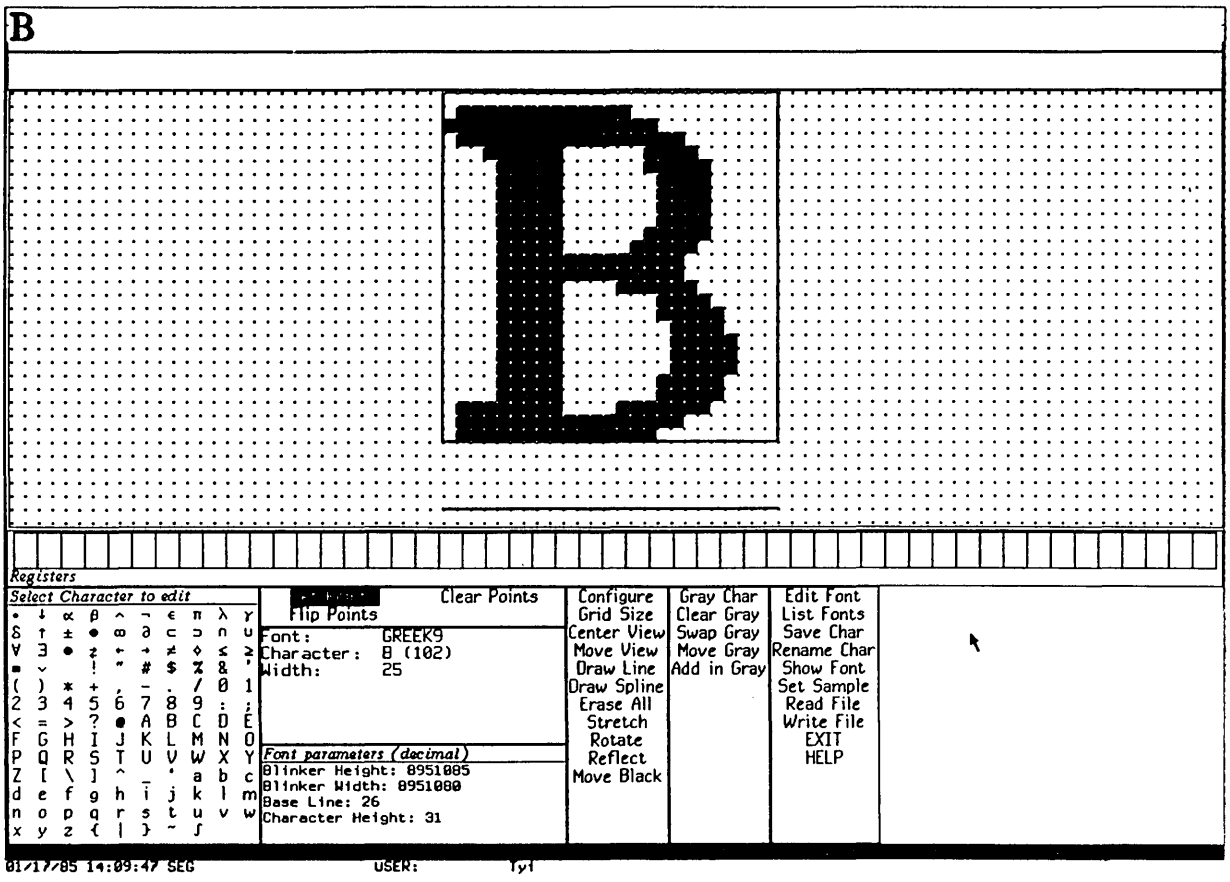


Figure 3. Wide Configuration

15.2.2 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 233.

15.3 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.

15.3.1 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.

15.3.2 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.

15.3.3 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.

15.3.4 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.

15.3.5 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.

16. 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 236.

16.1 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.

16.2 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 259. 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 259.

17. 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.

17.1 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.

17.2 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".

18. 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 "FED Registers".

18.1 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.

18.1.1 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.

18.1.2 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 240.

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.

18.2 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.

19. 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.

19.1 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.

19.2 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.

19.3 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.

20. 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.

20.1 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.

20.2 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.

Rotation	*Mouse command*
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.

20.3 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.

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* (\), through it.

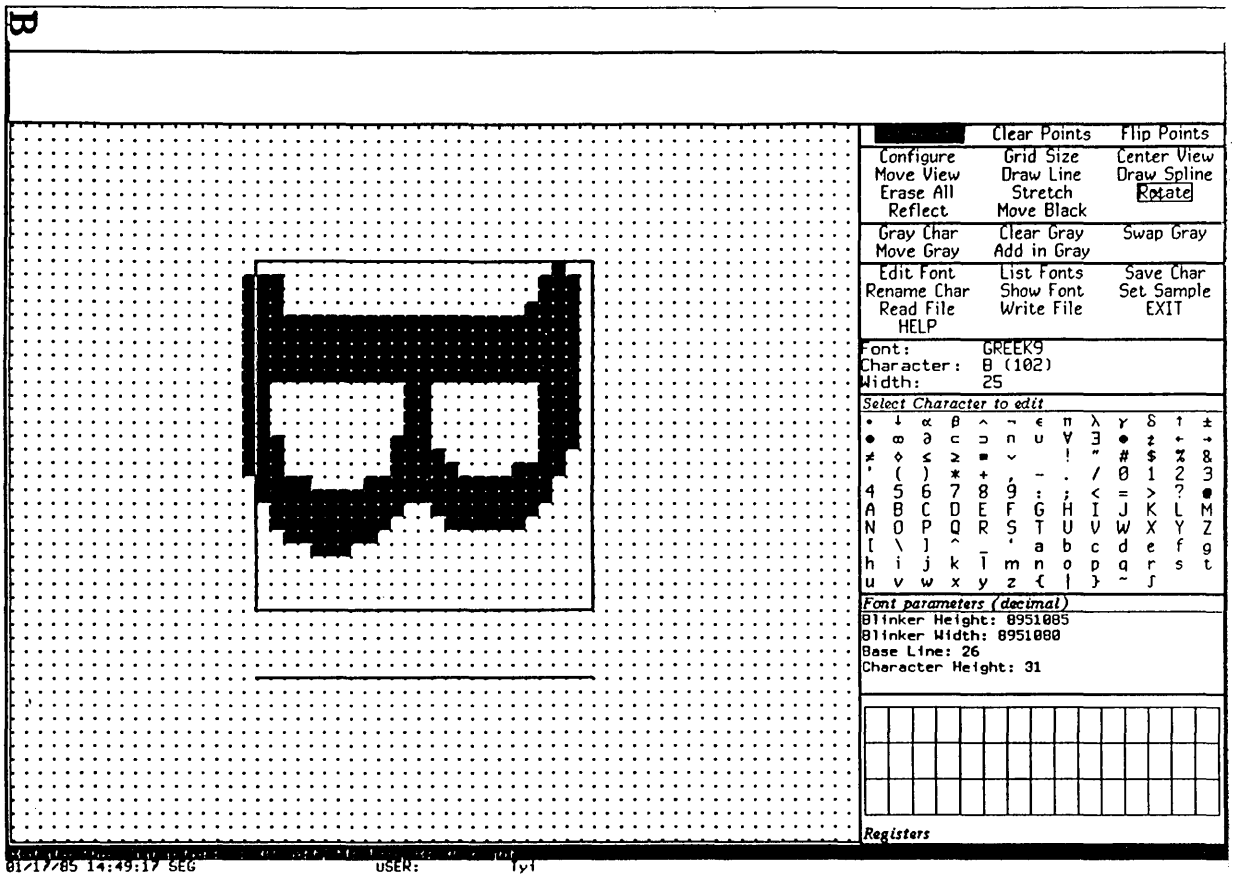


Figure 4. [Rotate (R)]

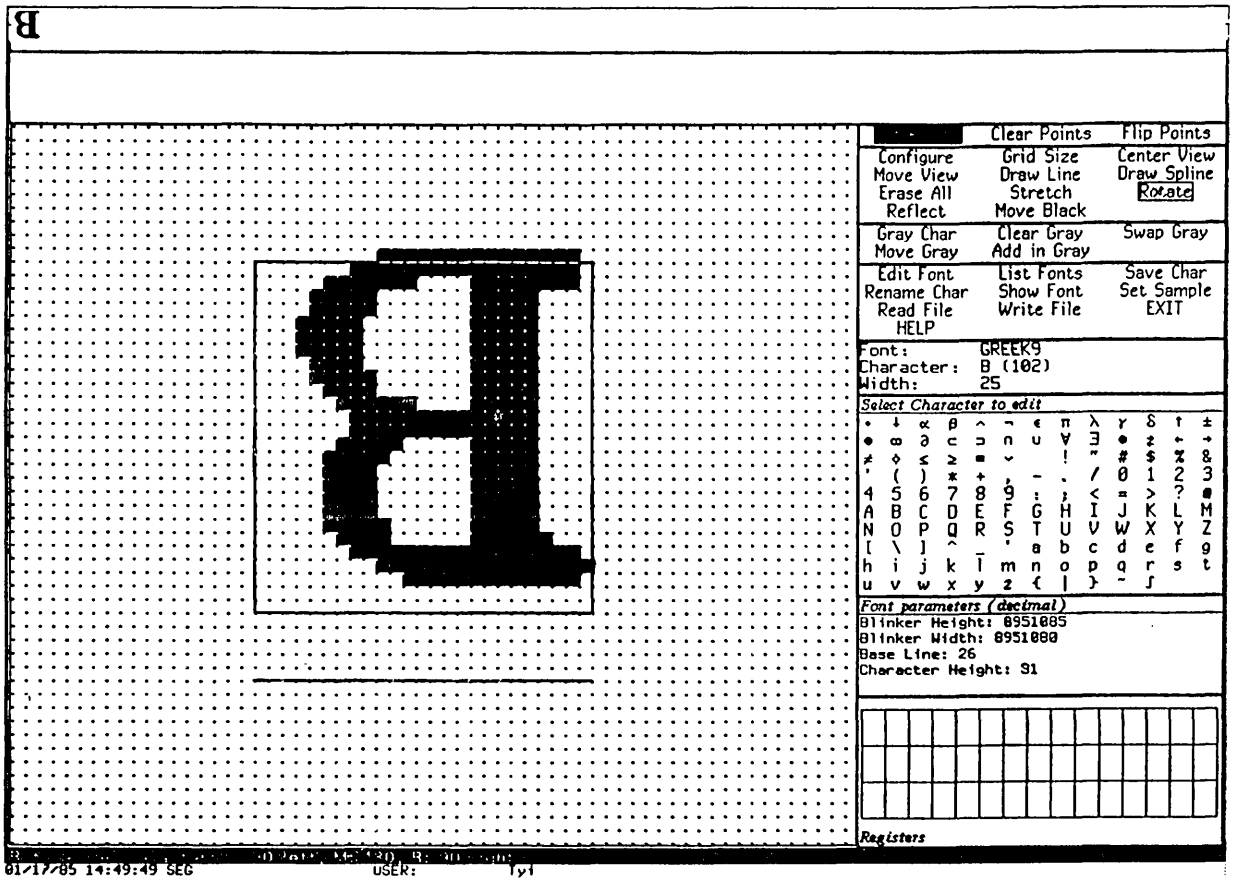


Figure 5. [Rotate (M)]

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.

20.4 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.

20.5 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.*

20.6 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 239.

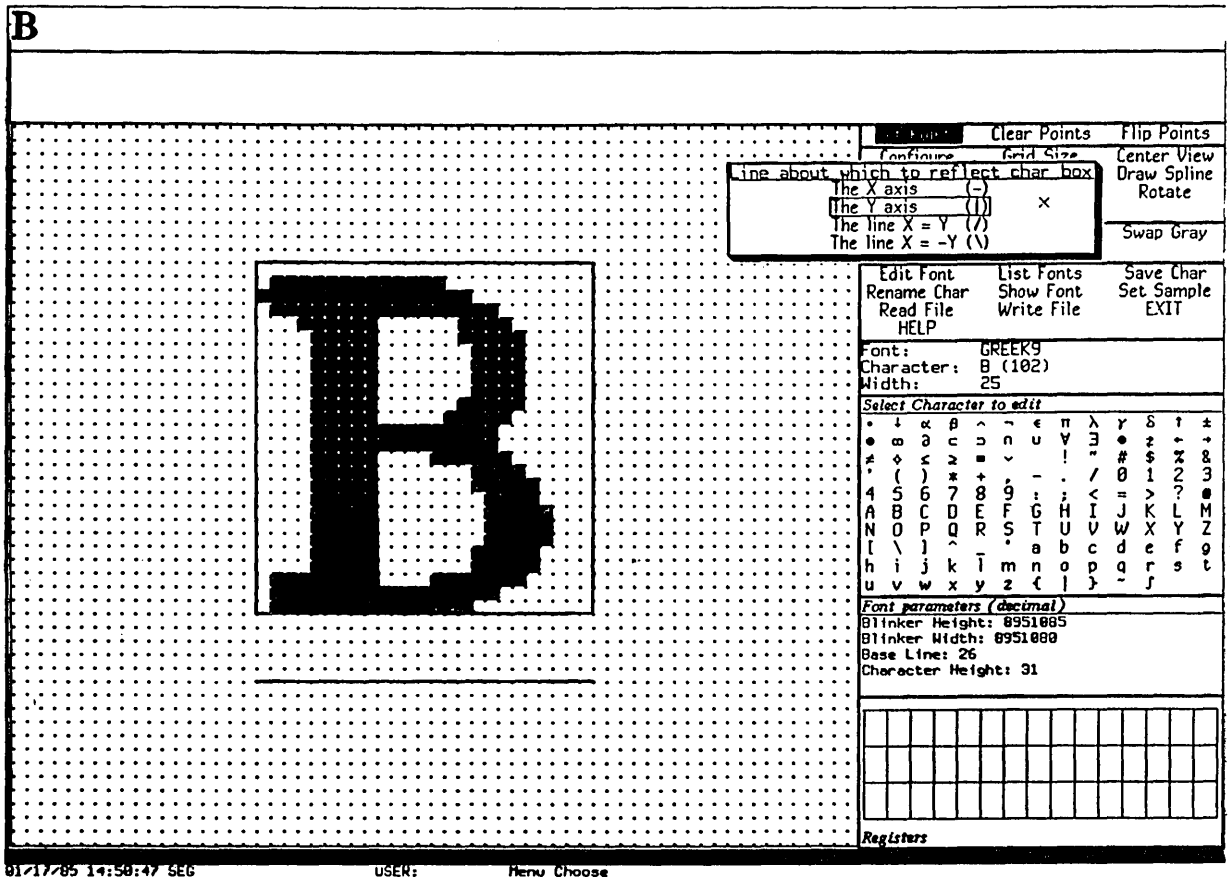


Figure 6. [Reflect]

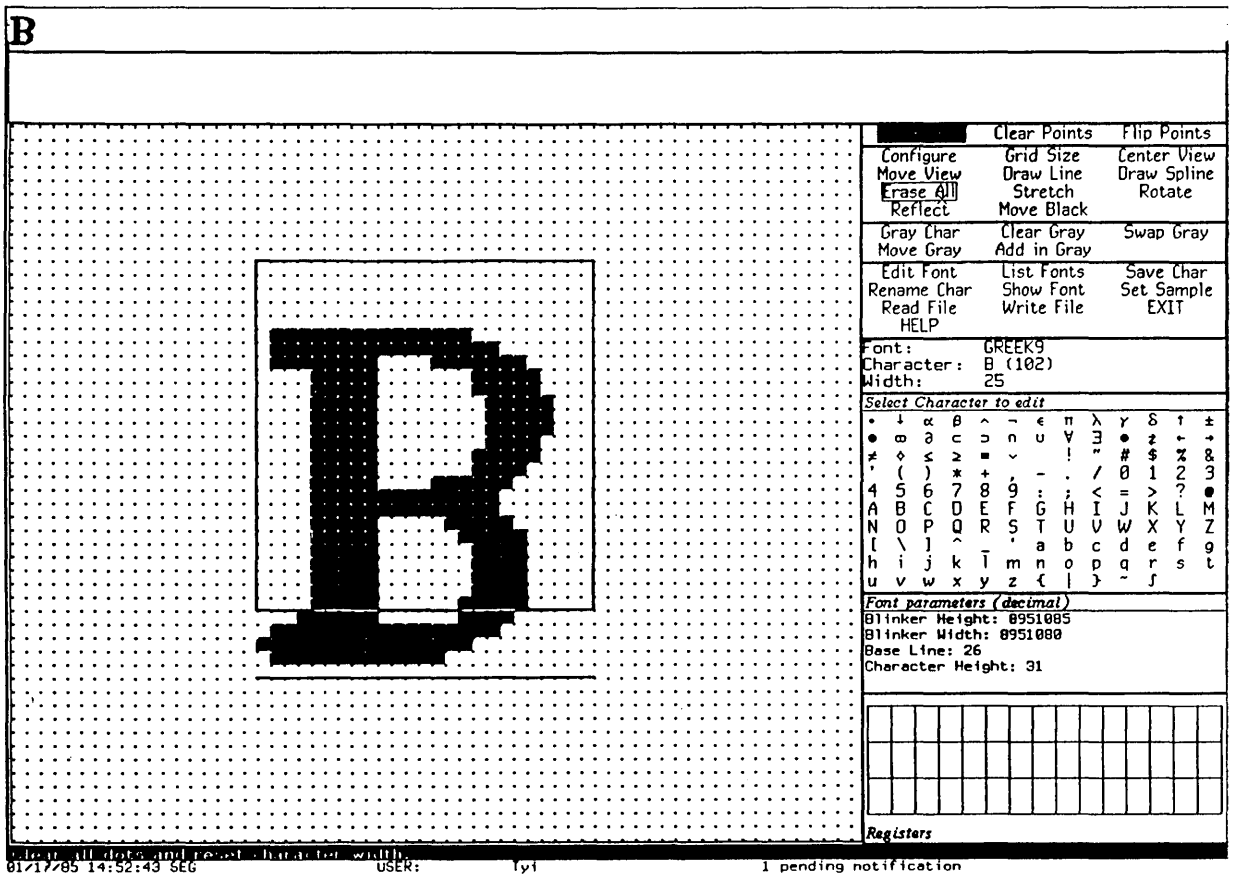


Figure 7. The X axis (-)

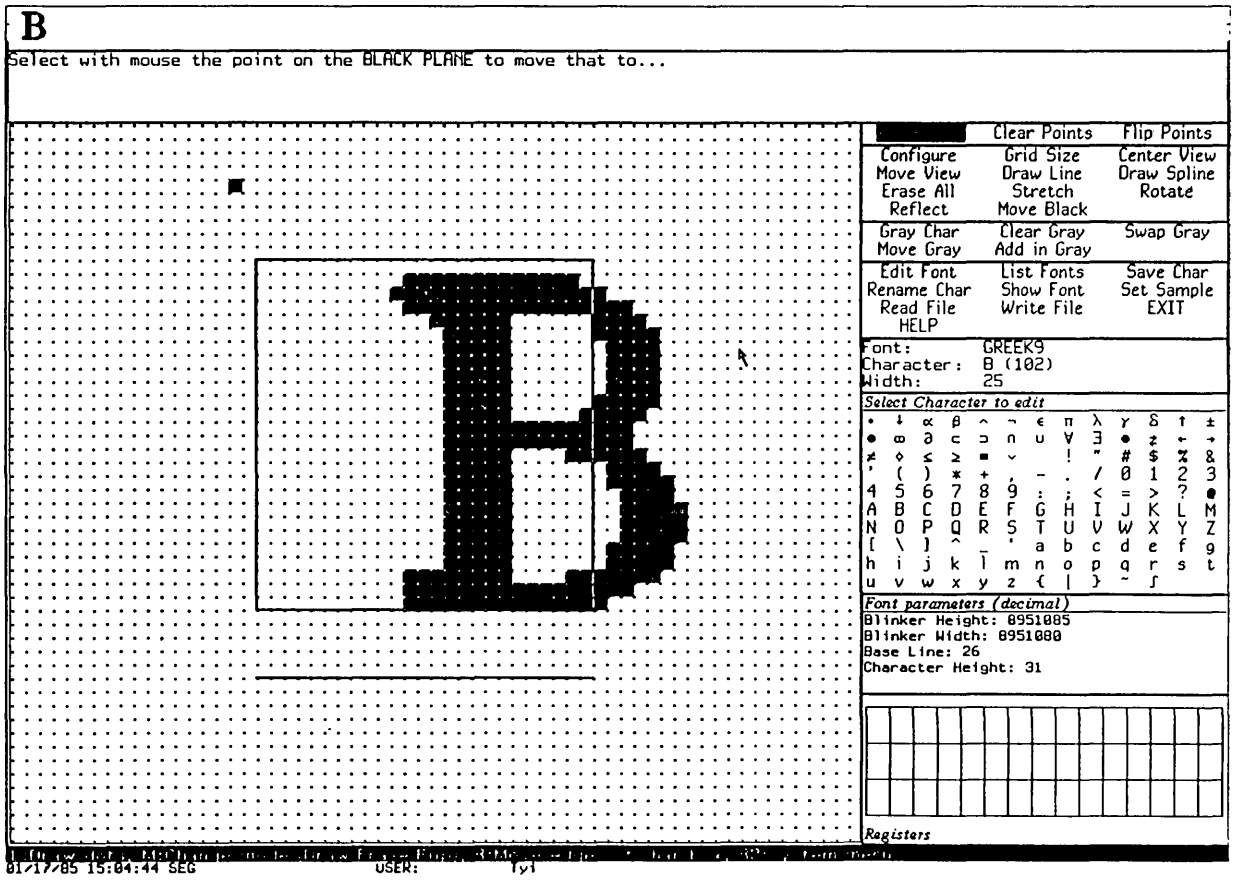


Figure 8. Moving the Drawing with [Move Black]

The relative orientation of the first and second points clicked on specifies whether you want to stretch or to contract.

20.6.1 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.

20.6.2 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.

20.6.3 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.

20.6.4 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.

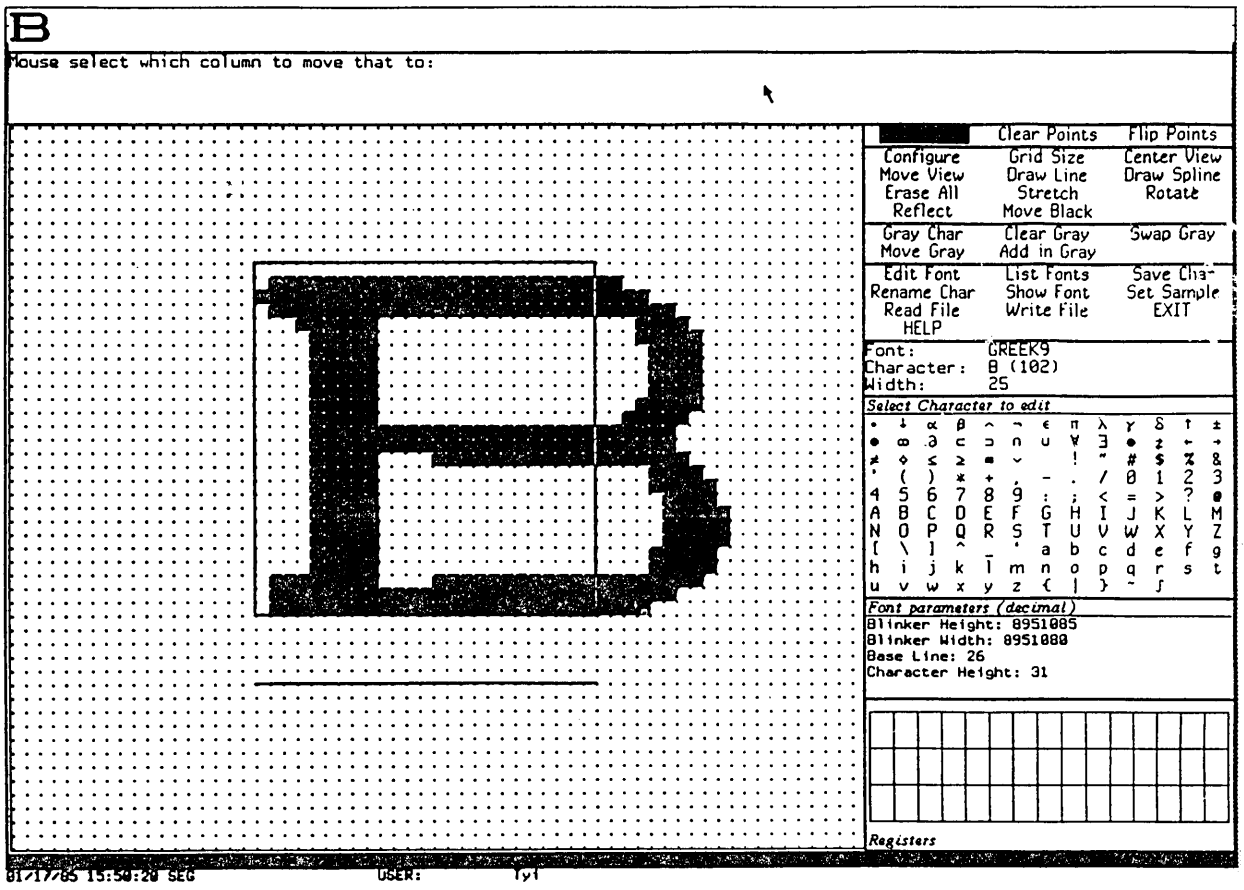


Figure 9. Stretching Horizontally

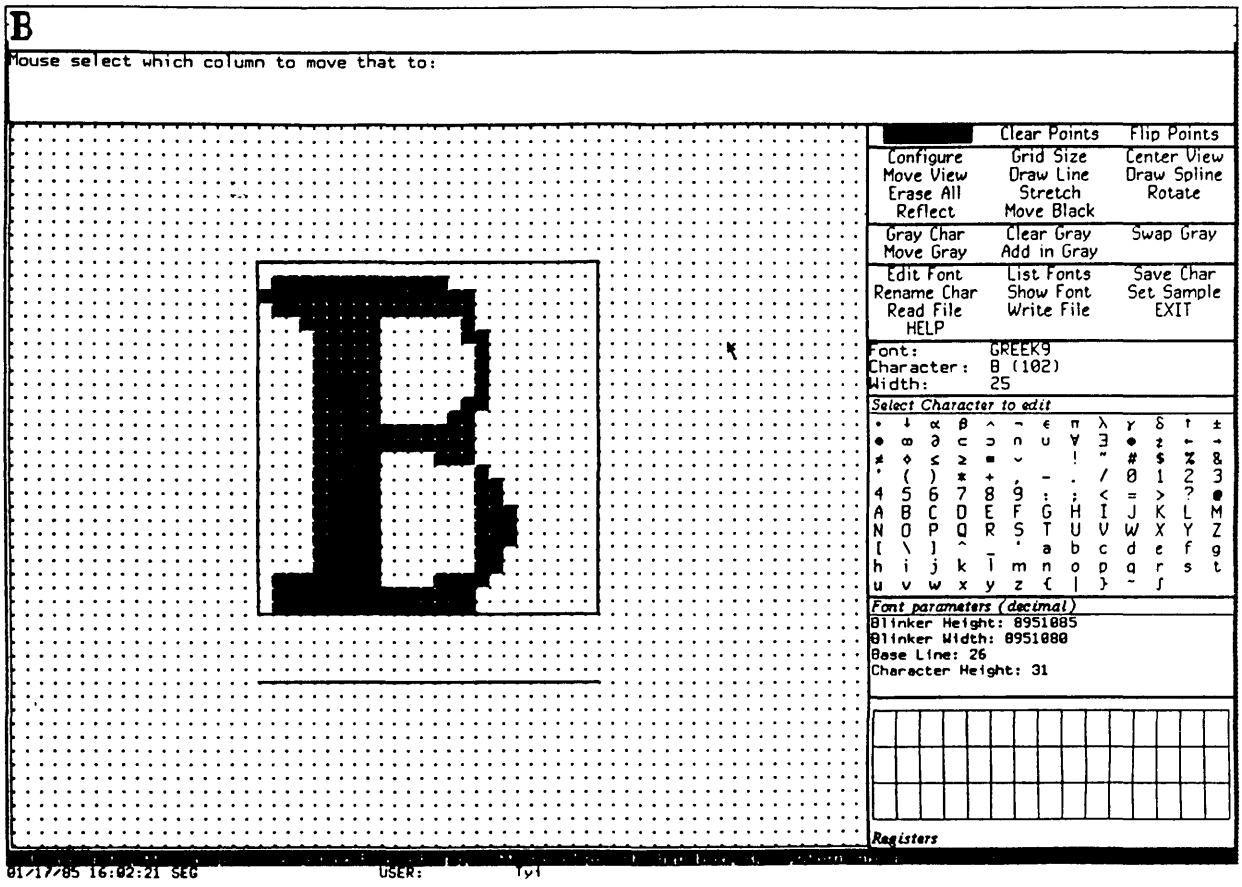


Figure 10. Contracting Horizontally

21. 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 v command from the keyboard. This last is often the most convenient, because you are then going to type the string itself.

22. 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.

22.1 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.

22.2 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.

22.3 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.

23. 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.

23.1 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 `A` (which queues a file to be acted on by a function). Apply the **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.

23.2 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 `W` 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.

24. 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.

24.1 Menu and Keyboard Commands

24.1.1 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 246.

[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.

24.1.2 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.

24.1.3 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.

24.1.4 Evaluating Forms From FED

FED uses the ESCAPE key to evaluate a Lisp form.

24.2 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.

\
: Turn the nonmouse cursor on, and move it one position up the screen. A numeric argument tells to move it other than one position. c-\ and m-\ mean 2 and 4 positions, respectively, and c-m-\ means 8.

/
: Same as \, but moves the nonmouse cursor down.

[
: Same as \, but moves the nonmouse cursor left.

]	Same as \, but moves the nonmouse cursor right.
.	When the nonmouse cursor is on, complement the black square under it.
REFRESH	Redraw the drawing pane. Useful in case of perceived problems.
c-REFRESH	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.
C	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. C accepts a character specification as a numeric argument.

24.3 Mouse Sensitivities

This section describes the result of clicking the mouse on various portions of the FED frame other than the command menus.

24.3.1 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.

24.3.2 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.

24.3.3 The Sample Pane

Clicking any button on the sample pane prompts for a new sample string.

24.3.4 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.

24.3.5 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.

24.3.6 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.

24.3.7 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.

PART III.

Hardcopy System

25. Printing and Hardcopy Commands

25.1 Commands for Producing Hardcopy

You can produce hardcopy using the System Menu, from the editor, from Zmail, from Dired in the editor, and from the file system editor. You can also get a hardcopy of your screen at any time.

In order for menu items, commands, and functions that refer to printing and hardcopy to work, your site must have a properly connected printing device. Printers are objects in the namespace database. See the section "Namespace System Printer Objects" in *Networks*.

25.1.1 Hardcopying From the System Menu

To produce hardcopy using the System Menu, click on [Hardcopy]. This pops up a menu that allows you to specify the pathname of the file to be hardcopied, the printer to send it to, and some options for format.

25.1.2 Hardcopying From Zmacs

You can hardcopy a region, a buffer, or a file from Zmacs.

Hardcopy Region (m-X)

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

Hardcopy Buffer (m-X)

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

Hardcopy File (m-X)

Sends a file to the local hardcopy device for printing.

Kill Or Save Buffers (m-X)

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.

25.1.3 Hardcopying From Zmail

You can hardcopy a single message or a collection of messages from Zmail.

Action	*Command*
One message	[Move / Hardcopy] Click right on its summary line then [Move / Hardcopy]
All messages in current sequence	[Map over / Move / Hardcopy]

In any of these commands you can use [Hardcopy (R)] to get a menu that permits you to specify the number of copies, the font, and which printer to use. The `Other` option in the list of printers allows you to specify an arbitrary printer, using either its pretty name or its namespace name. This printer becomes the selected printer, and remains in the menu for subsequent hardcopy commands.

25.1.4 Hardcopying From Dired

You can mark files to be hardcopied in Dired. When you exit from Dired, the files marked to be hardcopied are sent to the printer.

`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.

25.1.5 Hardcopying the Screen

You can get a hardcopy of what is displayed on your screen by pressing `FUNCTION Q`:

`Q`	Hardcopies the entire screen.
`c-Q`	Hardcopies the selected window.
`m-Q`	Hardcopies the entire screen, minus the status and mouse documentation lines.

25.1.6 Hardcopying From the File System Editor

You can invoke the system hardcopy menu from FSEdit. You click on `Hardcopy` in the menu of file operations invoked by clicking right on a file name.

25.2 Other Hardcopy Commands

25.2.1 Changing the Default Printer

When a site has more than one printer, one of the printers is specified as the site default hardcopy device. You can change the default in your init file to specify the printer that is most convenient for you. See the function **si:set-default-hardcopy-device**, page 271.

In the System Menu, using [Hardcopy] allows you to specify a different printer name; the printer name is mouse-sensitive.

25.2.2 Checking the Status of Hardcopy Devices

You can find out the status of a hardcopy device in Zmacs:

Show Hardcopy Status (m-X)

Show the status of a hardcopy device or all of them.

26. Customizing Hardcopy Facilities

You can specify the printer you want to use for hardcopying files and screen images in your init file.

si:set-default-hardcopy-device *name* *Function*

si:set-default-hardcopy-device specifies the printer to be used for all of the hardcopy commands except the screen copy command. *name* is a string specifying the device name. This is the real name of the printer, its **name** attribute not its **pretty-name**. For example: `caspian-sea` is the real name of the printer whose pretty name is `Caspian Sea`. (The valid set of device names are the **printer** objects in your namespace database.)

si:set-screen-hardcopy-device *name* *Function*

si:set-screen-hardcopy-device specifies the printer to be used for screen copies (by the `FUNCTION Q` command). *name* is a string specifying the device name. This is the real name of the printer, its **name** attribute not its **pretty-name**. For example: `caspian-sea` is the real name of the printer whose pretty name is `Caspian Sea`. (The valid set of device names are the **printer** objects in your namespace database.)

You can specify personal default fonts for each device in your init file.

si:*hardcopy-default-fonts* *Variable*

si:*hardcopy-default-fonts* is a variable whose value is an association list where each element specifies a device and a set of keyword/value pairs designating the font. The keywords are **:default-font** and **:header-font**.

For example:

```
(login-forms
  (setq si:*hardcopy-default-fonts*
        '(("Itasca" :default-font "Fix18B"))))
```

in your init file will specify Fix18B as the default font for the printer Itasca.

You can use `Show Font HELP` in the Lisp Listener or the List Fonts (`m-X`) command in Zmacs to get a list of all the fonts that are currently loaded into the Lisp environment. The **fonts** package contains the names of all fonts. Here is a list of some of the useful fonts:

fonts:cptfont	This is the default font, used for almost everything.
fonts:jess14	This is the default font in menus. It is a variable-width rounded font, slightly larger and more attractive than medfnt.

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 **cptfont**.
fonts:medfnb	This is a bold version of **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 *"More above"* and *"More below"*.
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 **hl10**, used for selected items in Choose Variable Values windows.

27. Hardcopy Functions

This chapter contains a list of some of the Lisp functions that refer to hardcopy facilities.

si:hardcopy-text-file *file-name device* &key *(page-headings t)* *(interpret-font-changes nil)* &allow-other-keys *Function*

si:hardcopy-text-file calls **si:hardcopy-from-stream** to do its work. Only one of the keywords **:fonts**, **:press-fonts**, or **:tv-fonts** should be given. The argument to these keywords is a list describing which fonts to use. The first element of the list corresponds to font 0 in the text file or font A in Zmacs. For example, to print a file on an LGP using the fonts TIMESROMAN10 and TIMESROMAN10B:

```
(si:hardcopy-text-file "f:>cwh>proposal.text" si:*default-hardcopy-device*
                       ':interpret-font-changes t
                       ':fonts '("TIMESROMAN10" "TIMESROMAN10B"))
```

The argument to the **:fonts** keyword is a list of device-specific fonts. If the output device is an LGP, it must be a list of strings representing LGP fonts. If the output device is a window, it must be the same as the argument to the **:tv-fonts** keyword. If the output device is the Dover, it must be the same as the argument to the **:press-fonts** keyword.

The argument to the **:tv-fonts** keyword is a list of symbols in the **fonts** package that represent fonts for the Lisp Machine screen. For example:

```
':tv-fonts '(fonts:tr10 fonts:tr10b)
```

If the hardcopy device is not a window, an attempt is made to translate the names of the fonts for the Lisp Machine screen into the names of fonts for the appropriate device.

The argument to the **:press-fonts** keyword is a list of triples representing Press fonts. For example:

```
':press-fonts '(("TIMESROMAN" "" 10.) ("TIMESROMAN" "B" 10.))
```

This keyword should be of interest only to those sites having a Xerox Dover printer.

The change to **si:hardcopy-from-stream** is for interim use at sites with a stand-alone Lisp Machine and LGP that use Zmacs as a text formatter. This also allows Japanese text to be printed on the LGP. Currently, the only way to print text files in multiple fonts is to call this function directly. No option exists for multiple fonts in the hardcopy menu.

si:hardcopy-from-stream *stream device* &rest *options* &key *(page-headings t)* *(interpret-font-changes nil)* *Function*

&allow-other-keys

si:hardcopy-from-stream recognizes font shift characters in files written by Zmacs. To print a file using multiple fonts, call **si:hardcopy-text-file** with **t** as the value of the keyword **:interpret-font-changes**, and also include one of the keywords **:fonts**, **:press-fonts**, or **:tv-fonts.**

si:make-hardcopy-stream *device* &key *fonts landscape-p* *Function*

Returns a hardcopy stream to the given device. This stream accepts the normal output stream messages, such as **:tyo**, **:string-out**, and some specific hardcopy messages outlined below.

Keyword	*Explanation*
:fonts	A list of device-dependent fonts, for example, **'("timesroman10" "timesroman10b")**
:landscape-p t	Causes the page to be printed with the long side horizontally.

Example,

```
(with-open-stream (stream (si:make-hardcopy-stream si:*default-hardcopy-device*))
  (send stream ':string-out "this is a test
of the hardcopy system."))
```

:eject-page of **si:make-hardcopy-stream** *Method*

Starts a fresh page. This is the same as **:tyo #\page.**

:set-font *new-font* of **si:make-hardcopy-stream** *Method*

Changes the current font of the stream. *new-font* can be either a number, which is an index into the stream's list of fonts, or a string, as given in the initial **:fonts** to **si:make-hardcopy-stream**. An error is given if the *new-font* is not in the list of fonts.

:set-cursorpos *x y* &optional *(units ':device)* of **si:make-hardcopy-stream** *Method*

Moves the place where printing occurs on the page to a new position. Unlike the Lisp Machine display, the 0,0 point of hardcopy streams is in the lower-left corner, the first (x) coordinate increasing toward the right of the page, the second (y) coordinate increasing toward the top of the page.

units specifies the format of x and y. *:device* means that the interpretation is device-dependent. *:micas* means x and y are in micas.

:show-rectangle *width height* of **si:make-hardcopy-stream** *Method*

Draws a filled-in rectangle on the page with the lower-left corner at the current position of size *width* x *height.*

Note: currently, *width* and *height* are always in device-dependent units. Use
(:method si:make-hardcopy-stream :convert-to-device-units) to convert from other units.

:convert-to-device-units *quantity units direction* of **si:make-hardcopy-stream** *Method*

Converts *quantity* in *units* into the corresponding quantity in device units. *direction* is either **:horizontal** or **:vertical**.

:show-line *to-x to-y* of **si:make-hardcopy-stream** *Method*

Draws a line on the page from the current position to the position designated by *to-x*, *to-y*.

Note: currently, the coordinates given to this message are different from those given to **:set-cursorpos**, in that the former do not add in the margins of the page, but the latter do. Use
(:method si:make-hardcopy-stream :un-relative-coordinates) to convert.

:un-relative-coordinates *x y* &optional *(units ':device)* of **si:make-hardcopy-stream** *Method*

Converts the point *x,y* as would be given to messages like **:set-cursorpos** that take coordinates relative to the page margins for use with messages like **:show-line** that take absolute coordinates.

Example:

```
(defun hardcopy-test (&optional (device si:*default-hardcopy-device*))
  (with-open-stream (stream (si:make-hardcopy-stream
                              device
                              ':fonts '("TIMESROMAN10" "TIMESROMAN20")))
    (send stream ':set-cursorpos (* 2 2540.) (* 2 2540.) ':micas)
    (send stream ':string-out "This is a test.")
    (send stream ':set-cursorpos (* 4 2540.) (* 7 2540.) ':micas)
    (send stream ':set-font 1)
    (send stream ':string-out "Yow!")
    (send stream ':set-cursorpos (* 3 2540.) (* 4 2540.) ':micas)
    (send stream ':show-rectangle
          (send stream ':convert-to-device-units (* 3 2540.) ':micas ':horizontal)
          (send stream ':convert-to-device-units (* 1 2540.) ':micas ':vertical))
    (let ((x-1 (* 3 2540.))
          (y-1 (* 6 2540.))
          (x-2 (* 6 2540.))
          (y-2 (* 8 2540.)))
      (multiple-value-bind (abs-x-1 abs-y-1)
          (send stream ':un-relative-coordinates x-1 y-1 ':micas)
        (multiple-value-bind (abs-x-2 abs-y-2)
            (send stream ':un-relative-coordinates x-2 y-2 ':micas)
          (send stream ':set-cursorpos x-1 y-1 ':micas)
          (send stream ':show-line abs-x-1 abs-y-2)
          (send stream ':set-cursorpos x-1 y-2 ':micas)
          (send stream ':show-line abs-x-2 abs-y-2)
          (send stream ':set-cursorpos x-2 y-2 ':micas)
          (send stream ':show-line abs-x-2 abs-y-1)
          (send stream ':set-cursorpos x-2 y-1 ':micas)
          (send stream ':show-line abs-x-1 abs-y-1))))))
```

Index

*
*
@* 37

)
)
Move Over) 67

!
!
! Dired command 153
Exclamation point (!) line continuation indicator 23, 54

#
#
@# 37

$
$
$ Dired command 152

+
+
+ flag in Zmacs 115

,
,
, Dired command 149

.
.
@. 37
. Font Editor drawing command 236

/
/
/ Font Editor command 261

1
1
Loop Indentor Example 1 163
Example 1 of Making Tables Using Keyboard Macros 206
Example 1 of Writing and Saving Keyboard Macros 201
Example 1 of Zmacs Notation Conventions 9
c-X 1 Zmacs command 130

2

Loop Indentor Example 2 163
Example 2 of Making Tables Using Keyboard Macros 206
Example 2 of Writing and Saving Keyboard Macros 202
Example 2 of Zmacs Notation Conventions 9
c-X 2 Zmacs command 129

3

Example 3 of Zmacs Notation Conventions 9
c-X 3 Zmacs command 129

4

c-X 4 Zmacs command 129

8

c-X 8 Zmacs command 129

;

Semicolon (;) comment indicator 171
c-X ; Zmacs command 172

=

@= 37
= Dired command 150
c-X = Zmacs command 47

>

@> 37

?

? Dired command 148
? Font Editor command 260
HELP ? Zmacs command 14

A

Example of a Search String for HELP A 46
Dired Abort 148
Abort At Top Level 42
ABORT Dired command 148
ABORT Font Editor command 261
Abort Patch (m-X) 191
ABORT Zmacs command 42
Warnings about file attribute lists 138
Zmacs Commands for Finding Out About Flavors 218
Zmacs Commands for Finding Out About Lisp 217

Send mail about patch 190
Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
Finding Out About Zmacs Commands 44
More HELP Commands for Finding Out About Zmacs Commands 46
Overview of Finding Out About Zmacs Commands 44
Finding Out About Zmacs Commands with HELP 44
Finding Out About Zmacs Variables 210
Accidental deletion 43
Active patches 186, 190
[Add in Gray] Font Editor menu item 240, 260
Add Patch Changed Definitions (m-X) 189
Add Patch Changed Definitions of Buffer (m-X) 188
Add Patch (m-X) 188
Add region to patch file 186
A Dired command 153
Adjusting the FED Display 255
Aligning Indentation in Zmacs 165
Undo all changes to buffer 121
[Erase All] Font Editor menu item 239, 243, 259
Clear all points 259
Motion Along One Nesting Level 65
Viewing and Altering a Character in the FED Character Box 237
Altering the FED Character Box 238
Alternative configuration 227
Fast Where Am I 47
Where Am I 47
Motion Among Top-level Expressions 66
Moving the Drawing Horizontally and/or Vertically in FED 255
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Copying a File Into Another 134
Examples of Copying a File Into Another 134
Inserting a Buffer Into Another Buffer 124
Any Extended Command 7
Appending, Prepending, and Inserting Text in Zmacs 124
Appending a Region to a Buffer 124
Appending a Region to a File 124
Append Next Kill 75
Append To Buffer 124
Append To File (m-X) Zmacs command 124
Dired Apply Function 153
Applying Arbitrary Functions to Files in Dired 153
Searching for Appropriate Commands 45
Method for Searching for Appropriate Zmacs Commands 45
Searching for Appropriate Zmacs Commands 45
Apropos 14
Variable Apropos 210
Variable Apropos Zmacs command 210
Applying Arbitrary Functions to Files in Dired 153
Echo Area 19
Zmacs Echo Area 19
Echo Area's Minibuffer 19
Quadruple Numeric Arg 24
Quick Arglist 47
Arglist (M-x) 48
Arglist (m-X) Zmacs command 48
Display argument list 47, 48

Defaults to Numeric Arguments 24
Echoing arguments 24
Example of Numeric Arguments 24
Numeric arguments 22, 24, 45
Overview of Numeric Arguments 24
Negative Numeric Arguments and Motion Commands 60
Numeric Arguments and the Motion Commands 60
Example of Negative Numeric Arguments with Motion Commands 60
Example of Numeric Arguments with Motion Commands 60
Positioning the Window Around a Definition 56
Moving Around in Dired 149
Assign key bindings 208
Associating a File with a Buffer 122
Association of buffers with files 30
Association of files with buffers 30
Init File Form: Balanced Quotation Marks and Asterisks 215
Atom Query Replace 104
Backspace Attribute 141
Backspace file attribute 141, 143
Base Attribute 141
Base file attribute 141, 143
Baseline Font Attribute 222
Character Height Font Attribute 221
Character Width Font Attribute 222
Char-exists Table Font Attribute 223
Fixed-width Font Attribute 222
Fonts Attribute 141
Left Kern Font Attribute 222
Lowercase Attribute 142
Lowercase file attribute 142, 143
Nofill Attribute 142
Nofill file attribute 142, 143
Patch-file Attribute 143
Patch-File file attribute 143
Tab-width Attribute 143
Tab-Width file attribute 143
Unknown attribute 138
Vsp Attribute 143
Vsp file attribute 143
File Attribute Checking 138
Buffer and File Attribute Descriptions 141
Attribute list 137
Describe Attribute List 149
Reparse Attribute List (m-X) Zmacs command 137
Update Attribute List (m-X) Zmacs command 138
Update Attribute List Query 141
Attribute lists 138
Warnings about file attribute lists 138
Attribute-manipulating Commands 137
Example of Attribute-manipulating Commands 138
Attributes 137
Blinker Width and *Blinker Height* Font Attributes 223
Buffer attributes 137
Character attributes 221
File attributes 137
Font attributes 221
Other Set Commands for File and Buffer. Attributes 140
Set commands for file and buffer attributes 143

Setting Buffer Attributes 141
Viewing File Attributes in Dired 149
Buffer and File Attributes in Zmacs 137
Attributes of TV Fonts 221
Init File Form: Auto Fill in Text Mode 214
Example of Filling Text with Auto Fill Minor Mode 195
Auto Fill Mode 142, 143, 196
Overview of Locating and Replacing Strings Automatically 102
Locating and Replacing Strings Automatically in Zmacs 102
Automatic drawing help 243
c-X A Zmacs command 124
HELP A Zmacs command 14, 45

B

B

B

@b 34
Getting Text Back 43
Finding Files That Have Not Been Backed up in Dired 152
Backspace Attribute 141
Backspace file attribute 141, 143
Set Backspace (m-X) Zmacs command 141, 143
Going Back to First Indented Character in Zmacs 165
File backup flag 153
Backward 26
Backward Character 61
Backward Kill Sentence 28
Backward Kill Sexp 28, 79
Backward Kill Word 28, 78
Backward List 65
Deleting Backward on the Line 82
Backward Page 69
Backward Paragraph 26, 68
Backward Sentence 26, 62
Backward Sexp 65
Erase backward to start of line 82
Backward up List 66
Kill Backward Up List (c-m-X) Zmacs command 79
Backward Word 26, 61
Init File Form: Balanced Quotation Marks and Asterisks 215
Scroll bar 255
Base 138
Base and Syntax Defaults 139
Base and Syntax Default Settings for Lisp 31, 120, 156, 170
Base Attribute 141
Base file attribute 141, 143
Baseline 222, 238
Baseline Font Attribute 222
Set Base (m-X) Zmacs command 141, 143
Font Basic Concepts 221
Font Editor Basic Concepts 227
Basic Text Formatting Commands 37
Basic Text Formatting Environments 34
Finding Files That Have Not Been Backed up in Dired 152
Beep 42
Goto Beginning 26, 70
Mark Beginning 92
Beginning/End of Buffer 70

Marking a Region From Here to Beginning of Buffer 92
Beginning of Definition 66
Beginning of Line 26, 63, 81
Move cursor to beginning of line 56
Protecting Files From Being Deleted in Dired 152
Protecting Files From Being Reaped in Dired 152
B Font Editor command 255
Query Replace Let Binding 104
Assign key bindings 208
Definition of Key Bindings 208
Extended command key bindings 46
Zmacs Key Bindings 208
Setting Key Bindings in Init Files 215
How Key Bindings Work: the Comtab 208
[Move Black] Font Editor menu item 238, 259
Black pane 239
Retrieving the Black Plane While Manipulating FED Registers 241
Deleting Blank Line in Zmacs 167
Inserting Blank Line in Zmacs 167
Delete Blank Lines 28
@blankspace 37
Blinker height 223
Blinker Width and *Blinker Height* Font Attributes 223
Blinker width 223
Blinker Width and *Blinker Height* Font Attributes 223
Boldface text 34
Zmacs Bolio Mode 157
Using Two Windows, Select Bottom 129
Bottom Edge of the FED Character Box 237
Altering the FED Character Box 238
Bottom Edge of the FED Character Box 237
Character box 227, 237, 238
Character Height of the FED Character Box 238
FED Character Box 227
Left and Right Edges of the FED Character Box 237
Top Edge of the FED Character Box 237
Using the mouse on the character box 238
Viewing and Altering a Character in the FED Character Box 237
What the Lines Mean in the FED Character Box 237
Size of boxes in the drawing pane 256, 259
Setting the Box Size in the FED Drawing Pane 256
Breaking a line 22
Appending a Region to a Buffer 124
Append To Buffer 124
Associating a File with a Buffer 122
Beginning/End of Buffer 70
Creating a Buffer 30, 31
Creating a Fundamental Mode Buffer 122
Current buffer 120
Current Zmacs Buffer 116
Editor Window's Buffer 17
Encrypting and Decrypting the Buffer 120
Evaluate and Replace Into Buffer 104
Format Buffer 38
Function-Specs-to-Edit-n buffer 13
Hardcopying the Buffer 119

Inserting a Buffer Into Another Buffer 124
Inserting a File Into a Buffer 124
Insert text from register into buffer 90
Marking a Region From Here to Beginning of Buffer 92
Marking a Region From Here to End of Buffer 92
Mode Line's *Buffer* 20
Motion with Respect to the Whole Buffer 70
Moving to end of buffer 70
Reading a File Into a New Buffer 120
Reading a File Into an Existing Buffer 120
Renaming the Buffer 119
Re-reading a File Into the Buffer 121
Select Buffer 30, 117
Select Default Previous Buffer 117
Selected buffer 116
Select Previous Buffer 117
The Editor Window and the Buffer 54
Undo all changes to buffer 121
View Buffer 119
Viewing a Buffer 119
Buffer and File Attribute Descriptions 141
Buffer and File Attributes in Zmacs 137
Zmacs Buffer and File Names 114
Buffer attributes 137
Other Set Commands for File and Buffer Attributes 140
Set commands for file and buffer attributes 143
Setting Buffer Attributes 141
Zmacs Buffer Commands 117
Writing the Buffer Contents to a File 121
Saving the Buffer Contents to the File 121
Buffer Contents with c-X c-F 31
Buffer Flags for Existing Files 114
Buffer Flags for New Files 115
Buffer History 116
Zmacs Buffer History 116
Inserting a Buffer Into Another Buffer 124
Init File Form: Ordering Buffer Lists 213
Add Patch Changed Definitions of Buffer (m-X) 188
Evaluate Buffer (m-X) Zmacs command 175
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
Evaluate Into Buffer (m-X) Zmacs command 175
Format Buffer (m-X) Zmacs command 33, 38
Hardcopy Buffer (m-X) Zmacs command 119, 267
Insert Buffer (m-X) Zmacs command 124
List Changed Definitions of Buffer (m-X) Zmacs command 183
Rename Buffer (m-X) Zmacs command 119
Revert Buffer (m-X) Zmacs command 121
View Buffer (m-X) Zmacs command 119
Buffer pointers 86
Buffers 30
Association of files with buffers 30
Changing Buffers 117
Commands to Mark Regions by Buffers 92
Destroying Buffers 123
Editing Buffers 118
Example of Listing Buffers 118
File buffers 122

List Buffers 118
Listing Buffers 117
Multiple buffers 114
Possibility Buffers 110
Reverting buffers 121, 122
Saving Buffers 120
Selecting, Listing, and Examining Zmacs Buffers 116
Support Buffers 109
Zmacs Commands for Finding Out About the State of Buffers 217
Creating and Saving Buffers and Files 30
Description of Creating and Saving Buffers and Files 30
Summary of Creating and Saving Buffers and Files 30
Manipulating Buffers and Files in Zmacs 113
Overview of Working with Buffers and Files in Zmacs 114
Working with Buffers and Files in Zmacs 114
Init File Form: Putting Buffers Into Current Package 213
Changing Case of Buffers in Zmacs 160
Comparing Files and Buffers in Zmacs 125
Edit Buffers (m-X) Zmacs command 118
Init File Form: Edit Buffers on c-X c-B 215
Init File Form: Edit Buffers on m-X 216
Mode Line's *Buffer-status* 21
Association of buffers with files 30
List Buffers Zmacs command 115
Built-in Customization Using Zmacs Minor Modes 195
c-X B Zmacs command 30, 31, 117

C

C

C

Help C 44
SELECT C 40
c-/ completion command 14
C-0 C-m-y 73
c-0 c-m-Y yank command 15
C-0 C-y 73
c-; Zmacs command 171
c-X c-; Zmacs command 173
c-= Zmacs command 47
HELP or c-? 14
c-? completion command 14
The Zmacs Edit Callers Commands 183
Calling the Last Keyboard Macro 200
Example of Calling the Last Keyboard Macro 200
Cancel last command 42
Cancel response 42
Canonical types 134
@caption 37
Carriage return 6
Changing Case and Indentation in Zmacs 159
Changing Case in Zmacs 160
Overview of Changing Case in Zmacs 160
Changing Case of Buffers in Zmacs 160
Changing Case of Regions in Zmacs 160
Changing Case of Words in Zmacs 160
c-A Zmacs command 26, 63, 81
Init File Form: Edit Buffers on c-X c-B 215

c-B Zmacs command 26, 61
c-X c-B Zmacs command 115, 118
c-C Font Editor command 260
Selecting a FED Character with the C Command 233
c-D Dired command 151
C Dired command 150
c-D Zmacs command 28, 43, 76
c-X c-D Zmacs command 132
Centering the Current Line in Zmacs 164
Centering the Drawing in FED 255
Center text 34
[Center View] Font Editor menu item 255, 259
c-E Zmacs command 26, 63, 81
Buffer Contents with c-X c-F 31
C Font Editor command 234, 261
c-F Zmacs command 26, 61
c-X c-F Zmacs command 31, 32
c-G Zmacs command 42
Add Patch Changed Definitions (m-X) 189
Evaluate Changed Definitions (m-X) Zmacs command 175
Add Patch Changed Definitions of Buffer (m-X) 188
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
List Changed Definitions of Buffer (m-X) Zmacs command 183
Change File Properties (m-X) Zmacs command 133
Undo all changes to buffer 121
Changing Buffers 117
Changing Case and Indentation in Zmacs 159
Changing Case in Zmacs 160
Overview of Changing Case in Zmacs 160
Changing Case of Buffers in Zmacs 160
Changing Case of Regions in Zmacs 160
Changing Case of Words in Zmacs 160
Changing File Properties in Dired 149
Changing the Default Printer 269
Changing the Properties of a File 133
Changing Window Size 129
Getting Things Into Gray with [Gray Char] 239
Backward Character 61
Contracting a character 259
Deleting the Current Character 76
Deleting the Last Character 76
Drawing a character 227
Editing a character 227
Forward Character 61
Motion by Character 60
RUBOUT Zmacs character 76
Selecting a FED Character by Creating a New Character 233
Stretching a character 259
Character attributes 221
Character box 227, 237, 238
Altering the FED Character Box 238
Bottom Edge of the FED Character Box 237
Character Height of the FED Character Box 238
FED Character Box 227
Left and Right Edges of the FED Character Box 237
Top Edge of the FED Character Box 237

Using the mouse on the character box 238
Viewing and Altering a Character in the FED Character Box 237
What the Lines Mean in the FED Character Box 237
Selecting a FED Character by Creating a New Character 233
Selecting a FED Character by Renaming Characters 234
Prefix character commands 6
Selecting a FED Character From the Character Select Menu 233
Selecting a FED Character From the [Show Font] Display 233
Character height 221, 227, 238
Character Height Font Attribute 221
Character Height of the FED Character Box 238
Selecting a Character in FED 233
Viewing and Altering a Character in the FED Character Box 237
Going Back to First Indented Character in Zmacs 165
Creating new characters 234
Deleting and Transposing Characters 76
Inserting Characters 22
Inserting Formatting Characters 23
Mirror imaging characters 243
Newline characters 22
Selecting a FED Character by Renaming Characters 234
Transposing Characters 76
Using the CONTROL key while drawing characters 235
Using the META key while drawing characters 235
Saving Characters and Pieces of Characters in FED Registers 241
Character Select menu 229
FED Character Select Menu 229
Selecting a FED Character From the Character Select Menu 233
Mousing on the FED Character Select Pane 263
Using the mouse in the character select pane 263
Transformations on Characters in FED 243
Saving Characters and Pieces of Characters in FED Registers 241
Drawing Characters in FED with the Mouse 235
Displaying Characters in the Font in FED 233
Merging Characters with the FED Gray Plane 240
Drawing characters with the mouse 235
Character width 222, 238
Character Width Font Attribute 222
Selecting a FED Character with the C Command 233
Char-exists table 223
Char-exists Table Font Attribute 223
[Gray Char] Font Editor menu item 239, 260
[Rename Char] Font Editor menu item 234, 260
[Save Char] Font Editor menu item 234, 260
File Attribute Checking 138
Checking the Status of Hardcopy Devices 269
Check Unbalanced Parentheses When Saving Variable 211
c-HELP V Zmacs command 210
Choosing the Other Window 130
c-X c-I Zmacs command 165
c-K Dired command 151
c-K Zmacs command 28, 81
Clean Directory (m-X) Zmacs command 136
Clear all points 259
[Clear Gray] Font Editor menu item 239, 260
Clear gray plane 239, 260

Clearing the Drawing in FED 243
CLEAR-INPUT Zmacs command 82
[Clear Points] Font Editor draw mode menu item 235
L2:Move to point mouse click 57
L:Move point mouse click 57
M2:Save/Kill/Yank mouse click 57
M:Mark thing mouse click 57
R2:System menu mouse click 57
R:Menu mouse click 57
Current meaning of mouse clicks 57
c-L Zmacs command 55
c-X c-L Zmacs command 160
c-m-; Zmacs command 171
c-m-? V Zmacs command 210
c-m-A Zmacs command 66
c-m-B Zmacs command 65
c-m-D Zmacs command 66
c-m-E Zmacs command 67
c-m-F Zmacs command 65
c-m-H Zmacs command 91
c-m-K Zmacs command 28, 79
Init File Form: c-m-L on the SQUARE Key 215
c-m-L Zmacs command 117
c-X c-m-L Zmacs command 117
c-m-N Zmacs command 65
c-m-O Zmacs command 166
c-m-P Zmacs command 65
c-m-Q Zmacs command 165
c-m-RUBOUT Zmacs command 28, 79
c-m-R Zmacs command 56
c-m-sh-E Zmacs command 175
c-m-SPACE Zmacs command 88
c-X c-m-SPACE Zmacs command 88
c-m-T Zmacs command 79
c-m-U Zmacs command 66
c-m-V Zmacs command 130
c-m-X 7
Kill Backward Up List (c-m-X) Zmacs command 79
C-0 C-m-y 73
c-m-Y yank command 15
c-0 c-m-Y yank command 15
c-m-[Zmacs command 66
c-m-] Zmacs command 67
c-m-@ Zmacs command 91
c-m-) Zmacs command 66
c-m-(Zmacs command 66
c-m-Z Zmacs command 175
c-m-\ Zmacs command 165
c-m-^ Zmacs command 166
c-N Dired command 149
c-N Zmacs command 26, 63, 81, 199
c-X c-N Zmacs command 63
c-0 c-Y yank command 15
Finding source code 170
Init File Form: White Space in Lisp Code 215
Commenting Lisp Code in Zmacs 171
Introduction to Locating Source Code in Zmacs 180
Overview of Commenting Lisp Code in Zmacs 171

Editing the source code of a function 12
Locating Source Code to Edit in Zmacs 180
Set Fill Column 196
Set Goal Column 63
Goal Column and the Motion Commands 63
Init File Form: Setting Goal Column for Real Line Commands 214
Setting the Lisp Comment Column in Zmacs 172
Default column position 63
! Dired command 153
$ Dired command 152
, Dired command 149
. Font Editor drawing command 236
/ Font Editor command 261
= Dired command 150
? Dired command 148
ABORT Dired command 148
ABORT Font Editor command 261
ABORT Zmacs command 42
A Dired command 153
Any Extended Command 7
Append To File (m-X) Zmacs command 124
Arglist (m-X) Zmacs command 48
B Font Editor command 255
c-/ completion command 14
c-0 c-m-Y yank command 15
c-; Zmacs command 171
c-= Zmacs command 47
c-? completion command 14
Cancel last command 42
c-A Zmacs command 26, 63, 81
c-B Zmacs command 26, 61
c-C Font Editor command 260
c-D Dired command 151
C Dired command 150
c-D Zmacs command 28, 43, 76
c-E Zmacs command 26, 63, 81
C Font Editor command 234, 261
c-F Zmacs command 26, 61
c-G Zmacs command 42
Change File Properties (m-X) Zmacs command 133
c-HELP V Zmacs command 210
c-K Dired command 151
c-K Zmacs command 28, 81
Clean Directory (m-X) Zmacs command 136
CLEAR-INPUT Zmacs command 82
c-L Zmacs command 55
c-m-; Zmacs command 171
c-m-? V Zmacs command 210
c-m-A Zmacs command 66
c-m-B Zmacs command 65
c-m-D Zmacs command 66
c-m-E Zmacs command 67
c-m-F Zmacs command 65
c-m-H Zmacs command 91
c-m-K Zmacs command 28, 79
c-m-L Zmacs command 117
c-m-N Zmacs command 65
c-m-O Zmacs command 166

c-m-P Zmacs command 65
c-m-Q Zmacs command 165
c-m-RUBOUT Zmacs command 28, 79
c-m-R Zmacs command 56
c-m-sh-E Zmacs command 175
c-m-SPACE Zmacs command 88
c-m-T Zmacs command 79
c-m-U Zmacs command 66
c-m-V Zmacs command 130
c-m-Y yank command 15
c-m-[Zmacs command 66
c-m-] Zmacs command 67
c-m-@ Zmacs command 91
c-m-) Zmacs command 66
c-m-(Zmacs command 66
c-m-Z Zmacs command 175
c-m-\ Zmacs command 165
c-m-^ Zmacs command 166
c-N Dired command 149
c-N Zmacs command 26, 63, 81, 199
c-O c-Y yank command 15
Compile Region (m-X) Zmacs command 94
COMPLETE completion command 14
Copy File (m-X) Zmacs command 134
c-O Zmacs command 167
c-P Dired command 149
c-P Zmacs command 26, 63, 81
Create Directory (m-X) Zmacs command 131
Create Link (m-X) Zmacs command 135
c-REFRESH Font Editor command 261
c-R Font Editor command 260
c-sh-A Zmacs command 47
c-sh-C Zmacs command 94
c-sh-D Zmacs command 47
c-sh-E Zmacs command 175
c-sh-V Zmacs command 48
c-SPACE Zmacs command 88
c-T Zmacs command 76
c-U Zmacs command 24
c-V Zmacs command 26, 55
c-W Font Editor command 260
c-W Zmacs command 28, 94
c-X 1 Zmacs command 130
c-X 2 Zmacs command 129
c-X 3 Zmacs command 129
c-X 4 Zmacs command 129
c-X 8 Zmacs command 129
c-X ; Zmacs command 172
c-X = Zmacs command 47
c-X A Zmacs command 124
c-X B Zmacs command 30, 31, 117
c-X c-; Zmacs command 173
c-X c-B Zmacs command 115, 118
c-X c-D Zmacs command 132
c-X c-F Zmacs command 31, 32
c-X c-I Zmacs command 165
c-X c-L Zmacs command 160
c-X c-m-L Zmacs command 117

c-X c-m-SPACE Zmacs command 88
c-X c-N Zmacs command 63
c-X c-O Zmacs command 28, 167
c-X c-P Zmacs command 92
c-X c-S Zmacs command 31, 121
c-X c-T Zmacs command 82
c-X c-U Zmacs command 160
c-X c-V Zmacs command 120
c-X c-W Zmacs command 32, 121
c-X c-X Zmacs command 88
c-X D Zmacs command 144
c-X E Zmacs command 200
c-X F Zmacs command 30, 196
c-X L Zmacs command 48
c-X O Zmacs command 130
c-X Q Zmacs command 202
c-X RUBOUT Zmacs command 28, 83
c-X S Zmacs command 30
c-X T Zmacs command 94
c-X V Zmacs command 119
c-X W Zmacs command 30
c-X [Zmacs command 26, 69
c-X] Zmacs command 26, 69
c-X Zmacs command 199
c-X) Zmacs command 199
c-X ^ Zmacs command 129
c-Y yank command 15
c-% Zmacs command 102
D Dired command 151
Deinstall Macro (m-X) Zmacs command 204
Delete File (m-X) Zmacs command 135
Describe Variable (m-X) Zmacs command 210
D Font Editor command 255, 260
Dired (m-X) Zmacs command 144
E Dired command 150
Edit Buffers (m-X) Zmacs command 118
Edit Directory (m-X) Zmacs command 144
E Font Editor command 243, 259
END completion command 14
END Dired command 148
End Kbd Macro Zmacs command 199
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Evaluate Buffer (m-X) Zmacs command 175
Evaluate Changed Definitions (m-X) Zmacs command 175
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
Evaluate Into Buffer (m-X) Zmacs command 175
Evaluate Region (m-X) Zmacs command 175
Example of the m-. Command 181
Extended Command 7
(fed) Lisp Listener command 225
F Font Editor command 260
Find File In Fundamental Mode (m-X) Zmacs command 122
Find File Zmacs command 31, 32
Finding the right command 45
Find Unbalanced Parentheses (m-X) Zmacs command 48
Format Buffer (m-X) Zmacs command 33, 38
Format File (m-X) Zmacs command 33, 38
Format Region (m-X) Zmacs command 33, 38

G Font Editor command 260
Hardcopy Buffer (m-X) Zmacs command 119, 267
HELP ? Zmacs command 14
HELP A Zmacs command 14, 45
HELP completion command 14
HELP C Zmacs command 14, 45
HELP Dired command 148
HELP D Zmacs command 14, 45
HELP Font Editor command 260
HELP L Zmacs command 14, 46
HELP SPACE Zmacs command 14
HELP U Zmacs command 14, 46
HELP V Zmacs command 14, 46, 210
HELP W Zmacs command 14
HELP W Zmacs command 46
H Font Editor command 259
Insert Buffer (m-X) Zmacs command 124
Insert File (m-X) Zmacs command 124
Install Command 209
Install Command (m-X) Zmacs command 209
Install Macro (m-X) Zmacs command 203
K Dired command 151
Kill Backward Up List (c-m-X) Zmacs command 79
LINE Zmacs command 164
Lisp Mode (m-X) Zmacs command 170
List Buffers Zmacs command 115
List Changed Definitions of Buffer (m-X) Zmacs command 183
List Definitions (m-X) Zmacs command 183
List Files (m-X) Zmacs command 131
List Fonts (m-X) Zmacs command 223, 271
List Variables (m-X) Zmacs command 210
m-; Zmacs command 171
m-< Zmacs command 26, 70
m-= Zmacs command 48
m-> Zmacs command 26, 70
m-A Zmacs command 26
m-B Zmacs command 26, 61
m-C Zmacs command 160
m-D Zmacs command 28, 78
m-ESCAPE Zmacs command 175
m-E Zmacs command 62
M Font Editor command 260
m-F Zmacs command 26, 61
m-H Zmacs command 92
m-K Zmacs command 28, 83
m-LINE Zmacs command 172
m-L Zmacs command 160
m-N Zmacs command 172
m-O Zmacs command 166
m-P Zmacs command 172
m-RUBOUT Zmacs command 28, 78
m-R Zmacs command 56
m-SCROLL Zmacs command 26, 56
m-sh-D Zmacs command 47
m-sh-E Zmacs command 175
m-S Zmacs command 164
m-T Zmacs command 78
m-U Zmacs command 160

m-V Zmacs command 26, 56
m-W Zmacs command 94
m-Y yank command 15, 51, 74
m-% Zmacs command 102
m-) Zmacs command 67
m-[Zmacs command 26, 68
m-] Zmacs command 26, 68
m-@ Zmacs command 91
m-Z Zmacs command 176
m-\ Zmacs command 28
m-^ Zmacs command 28, 166
Name Last Kbd Macro (m-X) Zmacs command 203
P Dired command 153, 268
Prepend To File (m-X) Zmacs command 124
Q Dired command 148
Q Font Editor command 260
Query Replace (m-X) Zmacs command 102
R Dired command 150
Reap File (m-X) Zmacs command 136
REFRESH Font Editor command 261
Rename Buffer (m-X) Zmacs command 119
Rename File (m-X) Zmacs command 133
Reparse Attribute List (m-X) Zmacs command 137
Repeat Last Minibuffer Command 51, 74
Replace String (m-X) Zmacs command 102
RETURN completion command 14
Revert Buffer (m-X) Zmacs command 121
RUBOUT Dired command 151
RUBOUT Zmacs command 28, 43
Save File Zmacs command 31
SCROLL Zmacs command 26
Selecting a FED Character with the C Command 233
Set Backspace (m-X) Zmacs command 141, 143
Set Base (m-X) Zmacs command 141, 143
Set Fonts (m-X) Zmacs command 141, 143
Set Lowercase (m-X) Zmacs command 142, 143
Set Nofill (m-X) Zmacs command 142, 143
Set Package (m-X) Zmacs command 138, 143
Set Patch File (m-X) Zmacs command 143
Set Tab Width (m-X) Zmacs command 143
Set Variable (m-X) Zmacs command 212
Set Visited File Name (m-X) Zmacs command 122
Set Vsp (m-X) Zmacs command 143
S Font Editor command 260
Show Directory (m-X) Zmacs command 132
Show File (m-X) Zmacs command 132
Source Compare Merge (m-X) Zmacs command 125
Source Compare (m-X) Zmacs command 125
SPACE completion command 14
SPACE Dired command 149
Trace (m-X) Zmacs command 48
U Dired command 151
Uncomment Region (m-X) Zmacs command 173
Update Attribute List (m-X) Zmacs command 138
Variable Apropos Zmacs command 210
V Dired command 150
V Font Editor command 253, 260
View Buffer (m-X) Zmacs command 119

View Directory (m-X) Zmacs command 132
View File (m-X) Zmacs command 132
View File Properties (m-X) Zmacs command 133
View Kbd Macro (m-X) Zmacs command 200
Write File Zmacs command 32
\ Font Editor command 261
? Font Editor command 260
H Font Editor command 255
Example of Finding Out What a Zmacs Command Does 44
Finding Out What an Extended Command Does 45
Finding Out What a Prefix Command Does 45
Finding Out What a Zmacs Command Does 44
Command history 43
Viewing the Editor Command History 73
Yanking in the command history 15
Extended command key bindings 46
FED Command List 259
Outside FED Command Menu 229
Command menus 228
Install Command (m-X) Zmacs command 209
Command Names 6
Commands 6
Attribute-manipulating Commands 137
Basic Text Formatting Commands 37
Cursor movement commands 26, 60
Delete commands 28
Dired Commands 145, 146
Editor Menu Commands 49
Evaluation commands 175
Example of Attribute-manipulating Commands 138
Example of Negative Numeric Arguments with Motion Commands 60
Example of Numeric Arguments with Motion Commands 60
Extended commands 6, 45
FED Keyboard-only Commands 261
FED Menu and Keyboard Commands 259
Finding Out About Zmacs Commands 44
General Information-giving Zmacs Commands 46
Goal Column and the Motion Commands 63
How to Use Formatting Commands 36
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: Setting Goal Column for Real Line Commands 214
Introduction to the Motion Commands 60
Introduction to Zmacs Commands 6
Introduction to Zmacs Extended Commands 7
Kill commands 28
List the last sixty commands 46
Method for Searching for Appropriate Zmacs Commands 45
More HELP Commands for Finding Out About Zmacs Commands 46
Motion Commands 60
Mouse-sensitive Zmacs commands 57
Names of commands 7, 45
Negative Numeric Arguments and Motion Commands 60
Numeric Arguments and the Motion Commands 60
Online documentation for commands 45
Other Hardcopy Commands 269
Other Region-related Commands 95
Overview of Finding Out About Zmacs Commands 44
Overview of Zmacs File Manipulation Commands 131

Prefix Commands 45
Prefix character commands 6
Printing and Hardcopy Commands 267
Region-manipulating Commands 94
Searching for Appropriate Commands 45
Searching for Appropriate Zmacs Commands 45
Set commands 143
The Zmacs Edit Callers Commands 183
The Zmacs Edit Definition Commands 180
The Zmacs List Definition Commands 182
Zmacs Buffer Commands 117
Zmacs File Manipulation Commands 131
Zmacs Format Commands 38
Zmacs Sorting Commands 112
Zmacs text formatting commands 37
Zmacs Window Commands 129
Compare/Merge Commands for Definitions 127
Other Set Commands for File and Buffer Attributes 140
Set commands for file and buffer attributes 143
Zmacs Commands for Finding Out About Flavors 218
Zmacs Commands for Finding Out About Lisp 217
Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
More HELP Commands for Finding Out About Zmacs Commands 46
Zmacs Commands for Formatting Text 33
Zmacs Commands for Interacting with Lisp 218
Commands for manipulating files 131
Commands for Producing Hardcopy 267
Commands to Mark Regions 91
Overview of Commands to Mark Regions 91
Commands to Mark Regions by Buffers 92
Commands to Mark Regions by Lisp Expressions 91
Commands to Mark Regions by Pages 92
Commands to Mark Regions by Paragraphs 92
Example of Commands to Mark Regions by Paragraphs 92
Commands to Mark Regions by Words 91
zwei: **command-store** 201
Dired Command Summary 146
Zmacs Help Command Summary 217
Finding Out About Zmacs Commands with HELP 44
Creating New Zmacs Commands with Keyboard Macros 199
Procedure for Creating Zmacs Commands with Keyboard Macros 199
Command table 208
Command tables 7
Introduction to Zmacs Command Tables 7
Setting the Lisp Comment Column in Zmacs 172
Semicolon (;) comment indicator 171
Commenting Lisp Code in Zmacs 171
Overview of Commenting Lisp Code in Zmacs 171
Indenting for Lisp Comment in Zmacs 171
Killing a Lisp Comment in Zmacs 171
Creating a New Indented Lisp Comment Line in Zmacs 172
Moving Down to Lisp Comment on Next Line in Zmacs 172
Moving up to Lisp Comment on Previous Line in Zmacs 172
Inserting and Removing Lisp Comments From Regions in Zmacs 173

Setting the Syntax for Symbolics Common Lisp 140
Example of a Source Compare 125
Source Compare 125, 131
Compare/Merge Commands for Definitions 127
Source Compare Installed Definition 127
Source Compare Merge 125
Source Compare Merge Installed Definition 128
Source Compare Merge (m-X) Zmacs command 125
Source Compare Merge Newest Definition 127
Source Compare Merge Saved Definition 127
Source Compare (m-X) Zmacs command 125
Source Compare Newest Definition 127
Source Compare Saved Definition 127
Comparing/Merging Current/Installed Versions 127
Comparing/Merging Current/Newest Versions 127
Comparing/Merging Current/Saved Versions 127
Comparing Files and Buffers in Zmacs 125
Comparing file versions 150
Comparing Recent Versions of Files in Dired 150
Compile Region 94
Compile Region (m-X) Zmacs command 94
Lisp Compiler Warnings 176
Compiling a Region 94
Compiling Lisp Programs in Zmacs 175
Evaluating and Compiling Lisp Programs in Zmacs 174
Overview of Evaluating and Compiling Lisp Programs in Zmacs 174
Dired Complement No Reap Flag 152
COMPLETE completion command 14
Completion 14
Introduction to Completion 14
c-/ completion command 14
c-? completion command 14
COMPLETE completion command 14
END completion command 14
HELP completion command 14
RETURN completion command 14
SPACE completion command 14
How Key Bindings Work: the Comtab 208
Standard comtab 208
Comtabs 7, 208
Font Basic Concepts 221
Font Editor Basic Concepts 227
Alternative configuration 227
Wide Configuration 227
FED Configuration and Drawing Transformation 259
Frame configurations 259
[Configure] Font Editor menu item 227, 256, 259
Deleting the List Containing the Current Lisp Expression 79
Creating Two Windows, Specifying Other Contents 129
Viewing and Editing File Contents in Dired 149
Displaying the Contents of a Directory 132
List contents of a directory 131
Retrieving the Contents of a FED Register 241
Writing the Buffer Contents to a File 121
Saving the Buffer Contents to the File 121
Buffer Contents with c-X c-F 31
Exclamation point (!) line continuation indicator 23, 54
Contracting a character 259

Contracting a Drawing Horizontally in FED 250
Contracting a Drawing Vertically in FED 250
Stretching and Contracting Drawings in FED 246
Using the CONTROL key while drawing characters 235
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
Entering Converse 40
:convert-to-device-units method of **si:make-hardcopy-stream** 275
Default printer for screen copies 271
Copy File 134
Copy File (m-X) Zmacs command 134
Copying a File Into Another 134
Examples of Copying a File Into Another 134
Copying and Renaming Files in Dired 150
Copying files 150
Correcting Typos 22
Count Lines Page 48
Count Lines Region 48
Setting Generation Retention Count on Files in Dired 152
c-0 Zmacs command 167
c-X c-0 Zmacs command 28, 167
c-P Dired command 149
fonts: **cptfont** font 223, 271
c-P Zmacs command 26, 63, 81
c-X c-P Zmacs command 92
FUNCTION c-Q 268
How to Create an Environment 33
Create Directory 131
Create Directory (m-X) Zmacs command 131
Create Link 135
Create Link (m-X) Zmacs command 135
Init File Form: Setting Find File Not to Create New Files 214
Creating a Buffer 30, 31
Creating a Directory 131
Example of Creating a Directory 131
Creating a File 32
Creating a Fundamental Mode Buffer 122
Creating and Saving Buffers and Files 30
Description of Creating and Saving Buffers and Files 30
Summary of Creating and Saving Buffers and Files 30
Selecting a FED Character by Creating a New Character 233
Creating a New Font in FED 230
Creating a New Indented Lisp Comment Line in Zmacs 172
Creating an Init File 213
Creating a Region 87
Creating a Region with Keystrokes 87
Creating a Region with the Mouse 87
Creating Links to Files 135
Creating new characters 234
Creating new fonts 230
Creating New Zmacs Commands with Keyboard Macros 199
Creating Two Windows, Specifying Other

Contents 129
Creating Two Windows with the Region in Top 129
Procedure for Creating Zmacs Commands with Keyboard Macros 199
c-REFRESH Font Editor command 261
c-R Font Editor command 260
c-sh-A Zmacs command 47
c-sh-C Zmacs command 94
c-sh-D Zmacs command 47
c-sh-E Zmacs command 175
c-sh-V Zmacs command 48
c-SPACE Zmacs command 88
c-X c-S Zmacs command 31, 121
c-T Zmacs command 76
c-X c-T Zmacs command 82
Draw a cubic spline 259
Comparing/Merging Current/Installed Versions 127
Comparing/Merging Current/Newest Versions 127
Comparing/Merging Current/Saved Versions 127
Current buffer 120
Deleting the Current Character 76
Deleting the Current Line 81
Centering the Current Line in Zmacs 164
Indenting Current Line in Zmacs 162
Deleting the Current Lisp Expression 79
Deleting the List Containing the Current Lisp Expression 79
Current meaning of mouse clicks 57
Init File Form: Putting Buffers Into Current Package 213
Current patch 190
Deleting the Current Sentence 83
Deleting the Current Word 78
Current Zmacs Buffer 116
Description of Moving the Cursor 26
Introduction to Moving the Cursor 26
Moving the cursor 57
Nonmouse cursor 235, 236, 261
Overview of Moving the Cursor 54
Relocate cursor 57
Summary of Moving the Cursor 26
The FED Nonmouse Cursor 236
Editor Window's Cursor and Point 17
Moving the Cursor in Zmacs 53
Summary of Cursor Movement 54
Cursor movement commands 26, 60
Move cursor to beginning of line 56
Moving the Cursor with the Mouse 57
Drawing Lines and Curves in FED 246
Built-in Customization Using Zmacs Minor Modes 195
Customizing Hardcopy Facilities 271
Customizing the Zmacs Environment 193
Overview of Customizing the Zmacs Environment 194
Introduction to Customizing Zmacs 194
Customizing Zmacs in Init Files 213
Introduction to Customizing Zmacs in Init Files 213
c-U Zmacs command 24
c-X c-U Zmacs command 160
c-V Zmacs command 26, 55
c-X c-V Zmacs command 120

c-W Font Editor command 260
c-W Zmacs command 28, 94
c-X c-W Zmacs command 32, 121
c-X 1 Zmacs command 130
c-X 2 Zmacs command 129
c-X 3 Zmacs command 129
c-X 4 Zmacs command 129
c-X 8 Zmacs command 129
c-X ; Zmacs command 172
c-X = Zmacs command 47
c-X A Zmacs command 124
c-X B Zmacs command 30, 31, 117
c-X c-; Zmacs command 173
Init File Form: Edit Buffers on c-X c-B 215
c-X c-B Zmacs command 115, 118
c-X c-D Zmacs command 132
Buffer Contents with c-X c-F 31
c-X c-F Zmacs command 31, 32
c-X c-I Zmacs command 165
c-X c-L Zmacs command 160
c-X c-m-L Zmacs command 117
c-X c-m-SPACE Zmacs command 88
c-X c-N Zmacs command 63
c-X c-O Zmacs command 28, 167
c-X c-P Zmacs command 92
c-X c-S Zmacs command 31, 121
c-X c-T Zmacs command 82
c-X c-U Zmacs command 160
c-X c-V Zmacs command 120
c-X c-W Zmacs command 32, 121
c-X c-X Zmacs command 88
c-X D Zmacs command 144
c-X E Zmacs command 200
c-X F Zmacs command 30, 196
c-X L Zmacs command 48
c-X O Zmacs command 130
c-X Q Zmacs command 202
c-X RUBOUT Zmacs command 28, 83
c-X S Zmacs command 30
c-X T Zmacs command 94
c-X V Zmacs command 119
c-X W Zmacs command 30
c-X [Zmacs command 26, 69
c-X] Zmacs command 26, 69
c-X Zmacs command 199
c-X c-X Zmacs command 88
c-X) Zmacs command 199
c-X ^ Zmacs command 129
C-0 C-y 73
c-Y yank command 15
c-0 c-Y yank command 15
Leaving Zmacs with c-Z 40
c-% Zmacs command 102
HELP C Zmacs command 14, 45

D

Help D 45
SELECT D 40
D Dired command 151
Encrypting and Decrypting the Buffer 120
Default column position 63
Default font 223, 271
:default-font keyword to **si:*hardcopy-default-fonts*** 271
Personal default fonts 271
Default major mode 198
Init File Form: Setting Default Major Mode 213
Default Pathnames in Dired 147
Select Default Previous Buffer 117
Move to Default Previous Point 88
Default printer 271
Changing the Default Printer 269
Default printer for screen copies 271
Base and Syntax Defaults 139
Base and Syntax Default Settings for Lisp 31, 120, 156, 170
Defaults to Numeric Arguments 24
Default syntax 140
One Window Default Variable 211
zwei: **define-keyboard-macro** 201
Defining an Interactive Keyboard Macro 202
Beginning of Definition 66
End of Definition 67
Mark Definition 91
Positioning the Window Around a Definition 56
Source Compare Installed Definition 127
Source Compare Merge Installed Definition 128
Source Compare Merge Newest Definition 127
Source Compare Merge Saved Definition 127
Source Compare Newest Definition 127
Source Compare Saved Definition 127
The Zmacs Edit Definition Commands 180
The Zmacs List Definition Commands 182
Editing the definition of a function 12
Definition of a Zmacs Keyboard Macro 199
Definition of a Zmacs Variable 210
Definition of Key Bindings 208
Definition of Zmacs Minor Modes 195
Compare/Merge Commands for Definitions 127
Add Patch Changed Definitions (m-X) 189
Evaluate Changed Definitions (m-X) Zmacs command 175
List Definitions (m-X) Zmacs command 183
Add Patch Changed Definitions of Buffer (m-X) 188
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
List Changed Definitions of Buffer (m-X) Zmacs command 183
zwei: **defmajor** 198
Deinstalling a Macro 204
Example of Installing and Deinstalling a Macro 204
Deinstall Macro (m-X) Zmacs command 204
Dired Delete 151
Delete Blank Lines 28
Delete commands 28
Protecting Files From Being Deleted in Dired 152
Delete File 135

Delete File (m-X) Zmacs command 135
Delete Forward 28, 76
Delete Horizontal Space 28
Delete Indentation 28
Deleting and Transposing Characters 76
Deleting and Transposing Lines 81
Introduction to Deleting and Transposing Lines 81
Deleting and Transposing Lisp Expressions 79
Introduction to Deleting and Transposing Lisp Expressions 79
Deleting and Transposing Text in Zmacs 71
Deleting and Transposing Words 78
Introduction to Deleting and Transposing Words 78
Deleting a Region 94
Deleting Backward on the Line 82
Deleting Blank Line in Zmacs 167
Deleting Files 135
Deleting Indentation in Zmacs 166
Deleting Multiple File Versions in Dired 151
Deleting Multiple Versions 135
Deleting Sentences 83
Introduction to Deleting Sentences 83
Deleting the Current Character 76
Deleting the Current Line 81
Deleting the Current Lisp Expression 79
Deleting the Current Sentence 83
Deleting the Current Word 78
Deleting the Last Character 76
Deleting the List Containing the Current Lisp Expression 79
Deleting the Previous Lisp Expression 79
Deleting the Previous Sentence 83
Deleting the Previous Word 78
Deleting Vs. Killing Text 72
Overview of Deleting Vs. Killing Text 72
Accidental deletion 43
Marking Files for Deletion in Dired 150
Large Deletions 43
Description of Zmacs Sentence Delimiters 62, 83
Descenders 238
Describe Attribute List 149
Describe Variable 210
Describe Variable At Point 48
Describe Variable (m-X) Zmacs command 210
Describing Zmacs Variables 210
Description of Creating and Saving Buffers and Files 30
Description of Erasing Text 28
Description of Motion by Lisp Expression 65, 79
Description of Moving the Cursor 26
Description of Zmacs Sentence Delimiters 62, 83
Buffer and File Attribute Descriptions 141
Description text 34
Destroying Buffers 123
Mouse as a graphic input device 227
Checking the Status of Hardcopy Devices 269
D Font Editor command 255, 260
Editing directories 144
Create Directory 131

Creating a Directory 131
Display Directory 132
Displaying the Contents of a Directory 132
Example of Creating a Directory 131
List contents of a directory 131
Listing Files in a Directory 131
Show Directory/View Directory 132
Show Directory/View Directory 132
Clean Directory (m-X) Zmacs command 136
Create Directory (m-X) Zmacs command 131
Edit Directory (m-X) Zmacs command 144
Show Directory (m-X) Zmacs command 132
View Directory (m-X) Zmacs command 132
Applying Arbitrary Functions to Files in Dired 153
Changing File Properties in Dired 149
Comparing Recent Versions of Files in Dired 150
Copying and Renaming Files in Dired 150
Default Pathnames in Dired 147
Deleting Multiple File Versions in Dired 151
Entering Dired 144
Finding Files That Have Not Been Backed up in Dired 152
Getting Out of Dired 147
Hardcopying From Dired 268
Loading a File in Dired 148
Marking Files for Deletion in Dired 150
Marking Files to Be Hardcopied in Dired 153
Moving Around in Dired 149
Online Documentation for Dired 148
Overview of Dired 144
Protecting Files From Being Deleted in Dired 152
Protecting Files From Being Reaped in Dired 152
Setting Generation Retention Count on Files in Dired 152
Viewing and Editing File Contents in Dired 149
Viewing File Attributes in Dired 149
Dired Abort 148
Dired Apply Function 153
! Dired command 153
$ Dired command 152
, Dired command 149
= Dired command 150
? Dired command 148
A Dired command 153
ABORT Dired command 148
C Dired command 150
c-D Dired command 151
c-K Dired command 151
c-N Dired command 149
c-P Dired command 149
D Dired command 151
E Dired command 150
END Dired command 148
HELP Dired command 148
K Dired command 151
P Dired command 153, 268
Q Dired command 148
R Dired command 150
RUBOUT Dired command 151
SPACE Dired command 149

U Dired command 151
V Dired command 150
Dired Commands 145, 146
Dired Command Summary 146
Dired Complement No Reap Flag 152
Dired Delete 151
The Dired Display 144
Updating the Dired Display 145
Dired Edit File 150
Dired Exit 148
Dired Hardcopy File 153, 268
Dired Help 148
Dired Menu 148, 149
Dired Mode in Zmacs 144
Dired move point 149
Dired (m-X) Zmacs command 144
Dired Next Undumped 153
Dired Reverse Undelete 151
Dired Srccom 150
Dired Undelete 151
Dired View File 150
Adjusting the FED Display 255
Selecting a FED Character From the [Show Font] Display 233
The Dired Display 144
Updating the Dired Display 145
Display] 55
FED display 225
Display argument list 47, 48
Display Directory 132
Displaying Characters in the Font in FED 233
Displaying previous keystrokes 46
Displaying the Contents of a Directory 132
Displaying the Next Possibility 110
Example of Displaying the Next Possibility 110
Displaying the Next Screen 55
Displaying the Previous Screen 56
Mousing on the FED List Fonts and Show Font Displays 263
Display text 34
Function Documentation 47
Long Documentation 47
Show Documentation 47
Status line documentation 255
Online documentation for commands 45
Online Documentation for Dired 148
Online documentation for prefixes 45
Mouse Documentation Line 57
Mouse Documentation Line in Zmacs 57
Entering Document Examiner 40
Example of Finding Out What a Zmacs Command Does 44
Finding Out What an Extended Command Does 45
Finding Out What a Prefix Command Does 45
Finding Out What a Zmacs Command Does 44
Introduction to Tag Tables and Search Domains 106
Tag Tables and Search Domains in Zmacs 106
Moving Rest of Line Down in Zmacs 166
Down Line 63, 81
Down List 66
Motion up and Down Nesting Levels 66

Down Real Line 26, 63, 81
Moving Down to Lisp Comment on Next Line in Zmacs 172
Draw a cubic spline 259
Draw a line 259
Moving the drawing 246, 259
Reflecting the drawing 243
Rotating the drawing 243, 259
Scrolling the drawing 255
Reflecting the drawing 259
Drawing a character 227
Using the CONTROL key while drawing characters 235
Using the META key while drawing characters 235
Drawing Characters in FED with the Mouse 235
Drawing characters with the mouse 235
. Font Editor drawing command 236
Automatic drawing help 243
Moving the Drawing Horizontally and/or Vertically in FED 255
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Contracting a Drawing Horizontally in FED 250
Stretching a Drawing Horizontally in FED 250
Drawing in FED 235
Centering the Drawing in FED 255
Clearing the Drawing in FED 243
Moving the Drawing in FED 246
Positioning the Drawing in FED 255
Move drawing in the gray plane 239, 260
Saving a Drawing into a FED Register 241
Drawing Lines and Curves in FED 246
Drawing pane 227, 262
FED Drawing Pane 227
Height and width of the drawing pane 256
Mousing on the FED Drawing Pane 262
Setting the Box Size in the FED Drawing Pane 256
Setting the Height and Width of the FED Drawing Pane 256
Size of boxes in the drawing pane 256, 259
Using the mouse in the drawing pane 262
Drawing Pane Menu 228
Reflecting Drawings in FED 243
Rotating Drawings in FED 243
Stretching and Contracting Drawings in FED 246
FED Configuration and Drawing Transformation 259
Contracting a Drawing Vertically in FED 250
Stretching a Drawing Vertically in FED 250
[Draw Line] Font Editor menu item 246, 259
Draw Mode Menu 228, 235, 262
Mousing on the FED Draw Mode Menu 262
Using the mouse in the draw mode menu 262
[Clear Points] Font Editor draw mode menu item 235
[Flip Points] Font Editor draw mode menu item 235
[Set Points] Font Editor draw mode menu item 235
[Draw Spline] Font Editor menu item 246, 259
c-X D Zmacs command 144
HELP D Zmacs command 14, 45

E

Entering Zmacs with SELECT E 12
SELECT E 12, 40
Echo Area 19
Zmacs Echo Area 19
Echo Area's Minibuffer 19
Echoing 19
Echoing arguments 24
Entering Zmacs with **ed** 12
ed function 12
Bottom Edge of the FED Character Box 237
Top Edge of the FED Character Box 237
Left and Right Edges of the FED Character Box 237
E Dired command 150
Edit Buffers (m-X) Zmacs command 118
Init File Form: Edit Buffers on c-X c-B 215
Init File Form: Edit Buffers on m-X 216
The Zmacs Edit Callers Commands 183
The Zmacs Edit Definition Commands 180
Edit Directory (m-X) Zmacs command 144
Dired Edit File 150
[Edit Font] Font Editor menu item 230, 260
Entering Zmacs with **zwei:** **edit-functions** 13
zwei: **edit-functions** function 13
Editing a character 227
Editing a File 32
Editing Buffers 118
Editing directories 144
Editing Existing Files 32
Viewing and Editing File Contents in Dired 149
Editing Lisp Programs in Zmacs 169
Introduction to Zmacs Major Editing Lisp Programs in Zmacs 170
Editing Modes 156
Editing the definition of a function 12
Editing the source code of a function 12
Locating Source Code to Edit in Zmacs 180
Entering File System Editor 40
Font Editor 219
Hardcopying From the File System Editor 268
Font Editor Basic Concepts 227
/ Font Editor command 261
ABORT Font Editor command 261
B Font Editor command 255
c-C Font Editor command 260
C Font Editor command 234, 261
c-REFRESH Font Editor command 261
c-R Font Editor command 260
c-W Font Editor command 260
D Font Editor command 255, 260
E Font Editor command 243, 259
F Font Editor command 260
G Font Editor command 260
HELP Font Editor command 260
H Font Editor command 259
M Font Editor command 260
Q Font Editor command 260
REFRESH Font Editor command 261
S Font Editor command 260

V Font Editor command 253, 260
\ Font Editor command 261
? Font Editor command 260
H Font Editor command 255
Viewing the Editor Command History 73
. Font Editor drawing command 236
[Clear Points] Font Editor draw mode menu item 235
[Flip Points] Font Editor draw mode menu item 235
[Set Points] Font Editor draw mode menu item 235
Overview of the Editor Menu 49
The Editor Menu 49
Editor Menu Commands 49
[Add in Gray] Font Editor menu item 240, 260
[Center View] Font Editor menu item 255, 259
[Clear Gray] Font Editor menu item 239, 260
[Configure] Font Editor menu item 227, 256, 259
[Draw Line] Font Editor menu item 246, 259
[Draw Spline] Font Editor menu item 246, 259
[Edit Font] Font Editor menu item 230, 260
[Erase All] Font Editor menu item 239, 243, 259
[EXIT] Font Editor menu item 260
[Gray Char] Font Editor menu item 239, 260
[Grid Size] Font Editor menu item 256, 259
[HELP] Font Editor menu item 260
[List Fonts] Font Editor menu item 230, 260
[Move Black] Font Editor menu item 238, 259
[Move Gray] Font Editor menu item 239, 260
[Move View] Font Editor menu item 255, 259
[Read File] Font Editor menu item 258, 260
[Reflect] Font Editor menu item 243, 259
[Rename Char] Font Editor menu item 234, 260
[Rotate] Font Editor menu item 243, 259
[Save Char] Font Editor menu item 234, 260
[Set Sample] Font Editor menu item 253, 260
[Show Font] Font Editor menu item 227, 233, 234, 260
[Stretch] Font Editor menu item 259
[Swap Gray] Font Editor menu item 239, 260
Using the mouse with [List Fonts] Font Editor menu item 263
Using the mouse with [Show Font] Font Editor menu item 234, 263
[Write File] Font Editor menu item 258, 260
Using the mouse with Font Editor menus 228
Setting Editor Variables in Init Files 213
Wraparound Lines in the Editor Window 54
- Zmacs Editor Window 17
Editor Window's Buffer 17
Editor Window's Cursor and Point 17
Editor Window's Typeout 17
The Editor Window and the Buffer 54
E Font Editor command 243, 259
:eject-page method of **si:make-hardcopy-stream** 274
Zmacs Electric Pl1 Mode 157
Init File Form: Electric Shift Lock in Lisp Mode 214
Retrieving History Elements 74
Using the Mouse on History Elements 74
Encrypting and Decrypting the Buffer 120
Goto End 26, 70
Mark End 92

END completion command 14
END Dired command 148
Ending a Keyboard Macro 200
End Kbd Macro Zmacs command 199
Marking a Region From Here to End of Buffer 92
Moving to end of buffer 70
End of Definition 67
End of Line 26, 63, 81
Entering and Leaving FED 225
Entering Converse 40
Entering Dired 144
Entering Document Examiner 40
Entering File System Editor 40
Entering Flavor Examiner 40
Entering Inspector 40
Entering Lisp 40
Entering Notifications 40
Entering Peek 40
Entering Terminal 40
Entering Zmacs 12, 40
Introduction to Entering Zmacs 12
Entering Zmacs with **ed** 12
Entering Zmacs with SELECT E 12
Entering Zmacs with the **Mouse** 12
Entering Zmacs with **zwei:edit-functions** 13
Entering Zmail 40
More Ways to Enter Minibuffer Responses 51
Using the mouse to enter Zmacs 12
Enumerate text 34
Customizing the Zmacs Environment 193
How to Create an Environment 33
Overview of Customizing the Zmacs Environment 194
Basic Text Formatting Environments 34
[Erase All] Font Editor menu item 239, 243, 259
Erase backward to start of line 82
Erasing text 76
Description of Erasing Text 28
Introduction to Erasing Text 28
Summary of Erasing Text 28
Error recovery 42
Escaping from prompts 42
Evaluate And Exit 175
Evaluate and Replace Into Buffer 104
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Evaluate Buffer (m-X) Zmacs command 175
Evaluate Changed Definitions (m-X) Zmacs command 175
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
Evaluate Into Buffer (m-X) Zmacs command 175
Evaluate Minibuffer 175
Evaluate Region 175
Evaluate Region (m-X) Zmacs command 175
Evaluate Region Verbose 175
Evaluating and Compiling Lisp Programs in Zmacs 174
Overview of Evaluating and Compiling Lisp Programs in

Zmacs 174
Evaluating Forms From FED 261
Evaluating Lisp Programs in Zmacs 174
Evaluation commands 175
Entering Document Examiner 40
Entering Flavor Examiner 40
Selecting, Listing, and Examining Zmacs Buffers 116
Mode Line Example 21
Text example 37
Loop Indentor Example 1 163
Example 1 of Making Tables Using Keyboard Macros 206
Example 1 of Writing and Saving Keyboard Macros 201
Example 1 of Zmacs Notation Conventions 9
Loop Indentor Example 2 163
Example 2 of Making Tables Using Keyboard Macros 206
Example 2 of Writing and Saving Keyboard Macros 202
Example 2 of Zmacs Notation Conventions 9
Example 3 of Zmacs Notation Conventions 9
Example of a Search String for HELP A 46
Example of a Source Compare 125
Example of a Tag Tables Replacement Operation 106
Example of Attribute-manipulating Commands 138
Example of Calling the Last Keyboard Macro 200
Example of Commands to Mark Regions by Paragraphs 92
Example of Creating a Directory 131
Example of Displaying the Next Possibility 110
Example of Filling Text with Auto Fill Minor Mode 195
Example of Finding Out What a Zmacs Command Does 44
Example of Installing and Deinstalling a Macro 204
Example of Listing Buffers 118
Example of Listing Variables by Matching a String 211
Example of Negative Numeric Arguments with Motion Commands 60
Example of Numeric Arguments 24
Example of Numeric Arguments with Motion Commands 60
Example of the m-. Command 181
Zmacs Notation Conventions and Examples 9
Examples of Copying a File Into Another 134
Example text 34
Exchange Lines 82
Exchange Regions 94
Exchange Sexps 79
Exchange Words 78
Exclamation point (!) line continuation indicator 23, 54
Reading a File Into an Existing Buffer 120
Buffer Flags for Existing Files 114
Editing Existing Files 32

Dired Exit 148
Evaluate And Exit 175
[EXIT] Font Editor menu item 260
Expanding Lisp Expressions in Zmacs 179
Deleting the Current Lisp Expression 79
Deleting the List Containing the Current Lisp Expression 79
Deleting the Previous Lisp Expression 79
Description of Motion by Lisp Expression 65, 79
· Motion by Lisp Expression 65
Reindenting Expression in Zmacs 165
Commands to Mark Regions by Lisp Expressions 91
Deleting and Transposing Lisp Expressions 79
Introduction to Deleting and Transposing Lisp Expressions 79
Motion Among Top-level Expressions 66
Transposing Lisp Expressions 79
Expanding Lisp Expressions in Zmacs 179
Parenthesizing Lisp Expressions in Zmacs 178
Extended Command 7
Any Extended Command 7
Finding Out What an Extended Command Does 45
Extended command key bindings 46
Extended commands 6, 45
Introduction to Zmacs Extended Commands 7
c-X E Zmacs command 200

F

F

SELECT F 40
Customizing Hardcopy Facilities 271
Fast Where Am I 47
Centering the Drawing in FED 255
Clearing the Drawing in FED 243
Contracting a Drawing Horizontally in FED 250
Contracting a Drawing Vertically in FED 250
Creating a New Font in FED 230
Displaying Characters in the Font in FED 233
Drawing in FED 235
Drawing Lines and Curves in FED 246
Entering and Leaving FED 225
Evaluating Forms From FED 261
Mouse Sensitivities in FED 262
Moving the Drawing Horizontally and/or Vertically in FED 255
Moving the Drawing in FED 246
Positioning the Drawing in FED 255
Reflecting Drawings in FED 243
Rotating Drawings in FED 243
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Selecting a Character in FED 233
Selecting a Font in FED 230
Stretching a Drawing Horizontally in FED 250
Stretching a Drawing Vertically in FED 250
Stretching and Contracting Drawings in FED 246
Transformations on Characters in FED 243
FED, the Subsystem 227
FED Character Box 227
Altering the FED Character Box 238
Bottom Edge of the FED Character Box 237
Character Height of the FED Character Box 238

F

Left and Right Edges of the FED Character Box 237
Top Edge of the FED Character Box 237
Viewing and Altering a Character in the FED Character Box 237
What the Lines Mean in the FED Character Box 237
Selecting a FED Character by Creating a New Character 233
Selecting a FED Character by Renaming Characters 234
Selecting a FED Character From the Character Select Menu 233
Selecting a FED Character From the [Show Font] Display 233
FED Character Select Menu 229
Mousing on the FED Character Select Pane 263
Selecting a FED Character with the C Command 233
FED Command List 259
Outside FED Command Menu 229
FED Configuration and Drawing Transformation 259
Adjusting the FED Display 255
FED display 225
FED Drawing Pane 227
Mousing on the FED Drawing Pane 262
Setting the Box Size in the FED Drawing Pane 256
Setting the Height and Width of the FED Drawing Pane 256
Mousing on the FED Draw Mode Menu 262
Reading FED Files 257
Reading and Writing FED Files 257
Writing FED Files 258
FED Font Parameters Menu 229
Mousing on the FED Font Parameters Menu 263
Getting Things Into the FED Gray Plane 239
Merging Characters with the FED Gray Plane 240
The FED Gray Plane 239
FED Gray Plane Menu Items 260
FED Keyboard-only Commands 261
(fed) Lisp Listener command 225
Mousing on the FED List Fonts and Show Font Displays 263
FED Menu and Keyboard Commands 259
FED Menus 228
The FED Nonmouse Cursor 236
FED Outside World Interface Menu Items 260
FED Prompt Pane 228
Retrieving the Contents of a FED Register 241
Saving a Drawing Into a FED Register 241
FED Register Pane 229
Mousing on the FED Register Pane 263
Retrieving the Black Plane While Manipulating FED Registers 241
Saving Characters and Pieces of Characters in FED Registers 241
FED Sample Pane 227
Mousing on the FED Sample Pane 262
The FED Sample String 253
FED Status Pane 229
Drawing Characters in FED with the Mouse 235
F Font Editor command 260
Add region to patch file 186
Appending a Region to a File 124
Changing the Properties of a File 133
Copy File 134
Creating a File 32
Creating an Init File 213
Delete File 135

Dired Edit File 150
Dired Hardcopy File 153, 268
Dired View File 150
Editing a File 32
Find File 30
Format File 38
Hardcopying a File 133
Install patch file 190
Naming a File 121
Prepending a Region to a File 124
Rename File 133
Renaming a File 133
Save File 30, 121
Saving a File 31
Saving the Buffer Contents to the File 121
Viewing a File 132
Viewing the Properties of a File 133
Visit File 120
Write File 30, 121
Writing the Buffer Contents to a File 121
Other Set Commands for File and Buffer Attributes 140
Set commands for file and buffer attributes 143
Backspace file attribute 141, 143
Base file attribute 141, 143
Lowercase file attribute 142, 143
Nofill file attribute 142, 143
Patch-File file attribute 143
Tab-Width file attribute 143
Vsp file attribute 143
File Attribute Checking 138
Buffer and File Attribute Descriptions 141
Warnings about file attribute lists 138
File attributes 137
Viewing File Attributes in Dired 149
Buffer and File Attributes in Zmacs 137
File backup flag 153
File buffers 122
Viewing and Editing File Contents in Dired 149
File flags 114
[Read File] Font Editor menu item 258, 260
[Write File] Font Editor menu item 258, 260
Supported file formats 258
Init File Form: Auto Fill in Text Mode 214
Init File Form: Balanced Quotation Marks and Asterisks 215
Init File Form: c-m-L on the SQUARE Key 215
Init File Form: Edit Buffers on c-X c-B 215
Init File Form: Edit Buffers on m-X 216
Init File Form: Electric Shift Lock in Lisp Mode 214
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: m-. on m-(L) 216
Init File Form: Ordering Buffer Lists 213
Init File Form: Putting Buffers Into Current Package 213
Init File Form: Setting Default Major Mode 213
Init File Form: Setting Find File Not to Create New Files 214
Init File Form: Setting Goal Column for Real Line

Commands 214
Init File Form: White Space in Lisp Code 215
Loading a File in Dired 148
Find File In Fundamental Mode (m-X) Zmacs command 122
Inserting a File Into a Buffer 124
Reading a File Into a New Buffer 120
Reading a File Into an Existing Buffer 120
Copying a File Into Another 134
Examples of Copying a File Into Another 134
Re-reading a File Into the Buffer 121
Overview of Zmacs File Manipulation Commands 131
Zmacs File Manipulation Commands 131
Append To File (m-X) Zmacs command 124
Copy File (m-X) Zmacs command 134
Delete File (m-X) Zmacs command 135
Format File (m-X) Zmacs command 33, 38
Insert File (m-X) Zmacs command 124
Prepend To File (m-X) Zmacs command 124
Reap File (m-X) Zmacs command 136
Rename File (m-X) Zmacs command 133
Set Patch File (m-X) Zmacs command 143
Show File (m-X) Zmacs command 132
View File (m-X) Zmacs command 132
Set Visited File Name (m-X) Zmacs command 122
Zmacs Buffer and File Names 114
Init File Form: Setting Find File Not to Create New Files 214
View File Properties 133
Changing File Properties in Dired 149
Change File Properties (m-X) Zmacs command 133
View File Properties (m-X) Zmacs command 133
Files 30
Association of buffers with files 30
Buffer Flags for Existing Files 114
Buffer Flags for New Files 115
Commands for manipulating files 131
Copying files 150
Creating and Saving Buffers and Files 30
Creating Links to Files 135
Customizing Zmacs in Init Files 213
Deleting Files 135
Description of Creating and Saving Buffers and Files 30
Editing Existing Files 32
Init Files 213
Init File Form: Setting Find File Not to Create New Files 214
Introduction to Customizing Zmacs in Init Files 213
Reading and Writing FED Files 257
Reading FED Files 257
Reading font files 258
Renaming Files 150
Setting Editor Variables in Init Files 213
Setting Key Bindings in Init Files 215
Setting Mode Hooks in Init Files 214
Summary of Creating and Saving Buffers and Files 30
Using the mouse with List Files 131
Writing FED Files 258
Writing font files 258
Comparing Files and Buffers in Zmacs 125

Marking Files for Deletion in Dired 150
Protecting Files From Being Deleted in Dired 152
Protecting Files From Being Reaped in Dired 152
Listing Files in a Directory 131
Applying Arbitrary Functions to Files in Dired 153
Comparing Recent Versions of Files in Dired 150
Copying and Renaming Files in Dired 150
Setting Generation Retention Count on Files in Dired 152
Manipulating Buffers and Files in Zmacs 113
Overview of Working with Buffers and Files in Zmacs 114
Working with Buffers and Files in Zmacs 114
List Files (m-X) Zmacs command 131
Finding Files That Have Not Been Backed up in Dired 152
Marking Files to Be Hardcopied in Dired 153
Association of files with buffers 30
Hardcopy files with multiple fonts 273
Entering File System Editor 40
Hardcopying From the File System Editor 268
fs:** **file-type-mode-alist Lisp variable 198
File Types and Zmacs Major Modes 198
File versions 114
Comparing file versions 150
Deleting Multiple File Versions in Dired 151
zwei: ***file-versions-kept*** variable 135
Associating a File with a Buffer 122
Find File Zmacs command 31, 32
Save File Zmacs command 31
Write File Zmacs command 32
Set Fill Column 196
Filling a Region 94
Example of Filling Text with Auto Fill Minor Mode 195
Init File Form: Auto Fill in Text Mode 214
Example of Filling Text with Auto Fill Minor Mode 195
Auto Fill Mode 142, 143, 196
Fill Paragraph 95
Set Fill Prefix 95
Fill Region 95
Find File 30
Find File In Fundamental Mode (m-X) Zmacs command 122
Init File Form: Setting Find File Not to Create New Files 214
Find File Zmacs command 31, 32
Finding Files That Have Not Been Backed up in Dired 152
Zmacs Commands for Finding Out About Flavors 218
Zmacs Commands for Finding Out About Lisp 217
Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
Finding Out About Zmacs Commands 44
More HELP Commands for Finding Out About Zmacs Commands 46
Overview of Finding Out About Zmacs Commands 44
Finding Out About Zmacs Commands with HELP 44
Finding Out About Zmacs Variables 210
Finding Out What an Extended Command Does 45
Finding Out What a Prefix Command Does 45
Finding Out What a Zmacs Command Does 44
Example of Finding Out What a Zmacs Command Does 44
Finding Out What You Have Typed 46

Finding source code 170
Finding the right command 45
Find Unbalanced Parentheses 48
Find Unbalanced Parentheses (m-X) Zmacs command 48
Finish Patch (m-X) 190
Going Back to First Indented Character in Zmacs 165
Fixed-width Font Attribute 222
Fixed-width fonts 222
Init File Form: Fixing White Space for Kill/Yank Commands 214
Dired Complement No Reap Flag 152
File backup flag 153
Modification flag 114
+ flag in Zmacs 115
File flags 114
Buffer Flags for Existing Files 114
Buffer Flags for New Files 115
Entering Flavor Examiner 40
Zmacs Commands for Finding Out About Flavors 218
[Flip Points] Font Editor draw mode menu item 235
Default font 223, 271
fonts:cptfont font 223, 271
Selecting a font 260
Baseline Font Attribute 222
Character Height Font Attribute 221
Character Width Font Attribute 222
Char-exists Table Font Attribute 223
Fixed-width Font Attribute 222
Left Kern Font Attribute 222
Font attributes 221
Blinker Width and *Blinker Height* Font Attributes 223
Font Basic Concepts 221
Selecting a FED Character From the [Show Font] Display 233
Mousing on the FED List Fonts and Show Font Displays 263
Font Editor 219
Font Editor Basic Concepts 227
] Font Editor command 261
[Font Editor command 261
/ Font Editor command 261
ABORT Font Editor command 261
B Font Editor command 255
C Font Editor command 234, 261
c-C Font Editor command 260
c-R Font Editor command 260
c-REFRESH Font Editor command 261
c-W Font Editor command 260

[[[[Font Editor command 261

]

] Font Editor command 261

]

F

F

F

D Font Editor command 255, 260
E Font Editor command 243, 259
F Font Editor command 260
G Font Editor command 260
H Font Editor command 259
HELP Font Editor command 260
M Font Editor command 260
Q Font Editor command 260
REFRESH Font Editor command 261
S Font Editor command 260
V Font Editor command 253, 260
\ Font Editor command 261
? Font Editor command 260
H Font Editor command 255
. Font Editor drawing command 236
[Clear Points] Font Editor draw mode menu item 235
[Flip Points] Font Editor draw mode menu item 235
[Set Points] Font Editor draw mode menu item 235
[Add in Gray] Font Editor menu item 240, 260
[Center View] Font Editor menu item 255, 259
[Clear Gray] Font Editor menu item 239, 260
[Configure] Font Editor menu item 227, 256, 259
[Draw Line] Font Editor menu item 246, 259
[Draw Spline] Font Editor menu item 246, 259
[Edit Font] Font Editor menu item 230, 260
[Erase All] Font Editor menu item 239, 243, 259
[EXIT] Font Editor menu item 260
[Gray Char] Font Editor menu item 239, 260
[Grid Size] Font Editor menu item 256, 259
[HELP] Font Editor menu item 260
[List Fonts] Font Editor menu item 230, 260
[Move Black] Font Editor menu item 238, 259
[Move Gray] Font Editor menu item 239, 260
[Move View] Font Editor menu item 255, 259
[Read File] Font Editor menu item 258, 260
[Reflect] Font Editor menu item 243, 259
[Rename Char] Font Editor menu item 234, 260
[Rotate] Font Editor menu item 243, 259
[Save Char] Font Editor menu item 234, 260
[Set Sample] Font Editor menu item 253, 260
[Show Font] Font Editor menu item 227, 233, 234, 260
[Stretch] Font Editor menu item 259
[Swap Gray] Font Editor menu item 239, 260
Using the mouse with [List Fonts] Font Editor menu item 263
Using the mouse with [Show Font] Font Editor menu item 234, 263
[Write File] Font Editor menu item 258, 260
Using the mouse with Font Editor menus 228
Reading font files 258
Writing font files 258
[Edit Font] Font Editor menu item 230, 260
[Show Font] Font Editor menu item 227, 233, 234, 260
Using the mouse with [Show Font] Font Editor menu item 234, 263

Creating a New Font in FED 230
Displaying Characters in the Font in FED 233
Selecting a Font in FED 230
Font Parameters menu 229
FED Font Parameters Menu 229
Mousing on the FED Font Parameters Menu 263
Using the mouse in the Font Parameters menu 263
:fonts 274
Attributes of TV Fonts 221
Creating new fonts 230
Fixed-width fonts 222
Hardcopy files with multiple fonts 273
Introduction to Fonts 221
Personal default fonts 271
Standard TV Fonts 223
Variable-width fonts 222
Mousing on the FED List Fonts and Show Font Displays 263
Fonts Attribute 141
fonts:cptfont font 223, 271
[List Fonts] Font Editor menu item 230, 260
Using the mouse with [List Fonts] Font Editor menu item 263
List Fonts (m-X) Zmacs command 223, 271
Set Fonts (m-X) Zmacs command 141, 143
@foot 37
Minibuffer Response Format 51
Format Buffer 38
Format Buffer (m-X) Zmacs command 33, 38
Zmacs Format Commands 38
Format File 38
Format File (m-X) Zmacs command 33, 38
Format Region 38
Format Region (m-X) Zmacs command 33, 38
Supported file formats 258
Producing Formatted Text 33
Format text 34
Text formatting 37
Inserting Formatting Characters 23
Basic Text Formatting Commands 37
How to Use Formatting Commands 36
Zmacs text formatting commands 37
Basic Text Formatting Environments 34
Introduction to Text Formatting in Zmacs 33
Zmacs Commands for Formatting Text 33
Init File Form: Auto Fill in Text Mode 214
Init File Form: Balanced Quotation Marks and Asterisks 215
Init File Form: c-m-L on the SQUARE Key 215
Init File Form: Edit Buffers on c-X c-B 215
Init File Form: Edit Buffers on m-X 216
Init File Form: Electric Shift Lock in Lisp Mode 214
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: m-. on m-(L) 216
Init File Form: Ordering Buffer Lists 213
Init File Form: Putting Buffers Into Current Package 213
Init File Form: Setting Default Major Mode 213
Init File Form: Setting Find File Not to Create New Files 214
Init File Form: Setting Goal Column for Real Line Commands 214

Evaluating Forms From FED 261
Init File Form: White Space in Lisp Code 215
Zmacs Fortran Mode 156
Forward 26
Delete Forward 28, 76
Forward Character 61
Forward List 65
Forward Page 69
Forward Paragraph 26, 68
Forward Sentence 62
Forward Sexp 65
Forward up List 66
Forward Word 26, 61
Scale fraction 239
Frame configurations 259
Protecting Files From Being Deleted in Dired 152
Protecting Files From Being Reaped in Dired 152
Hardcopying From Dired 268
Evaluating Forms From FED 261
Marking a Region From Here to Beginning of Buffer 92
Marking a Region From Here to End of Buffer 92
Inserting and Removing Lisp Comments From Regions in Zmacs 173
Selecting a FED Character From the Character Select Menu 233
Hardcopying From the File System Editor 268
Selecting a FED Character From the [Show Font] Display 233
Hardcopying From the System Menu 267
Hardcopying From Zmacs 267
Hardcopying From Zmail 268
fs:file-type-mode-alist Lisp variable 198
Dired Apply Function 153
ed function 12
Editing the definition of a function 12
Editing the source code of a function 12
note-private-patch function 192
si:hardcopy-from-stream function 274
si:hardcopy-text-file function 273
si:make-hardcopy-stream function 274
si:set-default-hardcopy-device function 271
si:set-screen-hardcopy-device function 271
zwei:edit-functions function 13
FUNCTION c-Q 268
Function Documentation 47
FUNCTION m-Q 268
FUNCTION Q 268, 271
Hardcopy Functions 273
Function-Specs-to-Edit-n buffer 13
Applying Arbitrary Functions to Files in Dired 153
Zmacs Fundamental Mode 156
Creating a Fundamental Mode Buffer 122
Find File In Fundamental Mode (m-X) Zmacs command 122
c-X F Zmacs command 30, 196

G

General Information-giving Zmacs Commands 46
Setting Generation Retention Count on Files in Dired 152
Getting Help in Zmacs 41
Getting Out of Dired 147
Getting Out of Keystroke Prefixes 42
Getting Out of Minibuffer Prompts 42
Getting Out of Prefixes and Prompts 42
Getting Out of Trouble 42
Overview of Getting Out of Trouble 42
Getting Started in Zmacs 11
Getting Text Back 43
Getting Things Into Gray with [Gray Char] 239
Getting Things Into Gray with [Swap Gray] 239
Getting Things Into the FED Gray Plane 239
G Font Editor command 260
Making Global Replacements in Zmacs 102
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
ibase global variable 141, 143
zwei:*set-attribute-update-list* global variable 141, 143
Set Goal Column 63
Goal Column and the Motion Commands 63
Init File Form: Setting Goal Column for Real Line Commands 214
Going Back to First Indented Character in Zmacs 165
Goto Beginning 26, 70
Goto End 26, 70
Mouse as a graphic input device 227
Getting Things Into Gray with [Swap Gray] 239
Getting Things Into Gray with [Gray Char] 239
[Gray Char] Font Editor menu item 239, 260
[Add in Gray] Font Editor menu item 240, 260
[Clear Gray] Font Editor menu item 239, 260
[Move Gray] Font Editor menu item 239, 260
[Swap Gray] Font Editor menu item 239, 260
Gray plane 228, 239
Clear gray plane 239, 260
Getting Things Into the FED Gray Plane 239
Merging Characters with the FED Gray Plane 240
Move drawing in the gray plane 239, 260
The FED Gray Plane 239
Gray Plane Menu 228
FED Gray Plane Menu Items 260
Getting Things Into Gray with [Gray Char] 239
Getting Things Into Gray with [Swap Gray] 239
[Grid Size] Font Editor menu item 256, 259
Grow Window 129

H

Marking Files to Be Hardcopied in Dired 153
Commands for Producing Hardcopy 267
Hardcopy Buffer (m-X) Zmacs command 119, 267
Other Hardcopy Commands 269
Printing and Hardcopy Commands 267
:default-font keyword to **si:** ***hardcopy-default-fonts*** 271

:header-font keyword to **si:** ***hardcopy-default-fonts*** 271
si: ***hardcopy-default-fonts*** variable 271
Checking the Status of Hardcopy Devices 269
Customizing Hardcopy Facilities 271
Dired Hardcopy File 153, 268
Hardcopy files with multiple fonts 273
si: **hardcopy-from-stream** function 274
Hardcopy Functions 273
Hardcopying a File 133
Hardcopying a Region 94
Hardcopying From Dired 268
Hardcopying From the File System Editor 268
Hardcopying From the System Menu 267
Hardcopying From Zmacs 267
Hardcopying From Zmail 268
Hardcopying the Buffer 119
Hardcopying the Screen 268
Hardcopy System 265
si: **hardcopy-text-file** function 273
[Hardcopy] Zmail menu item 268
Finding Files That Have Not Been Backed up in Dired 152
Finding Out What You Have Typed 46
:header-font keyword to **si:*hardcopy-default-fonts*** 271
Text heading 37
Blinker height 223
Character height 221, 227, 238
Line height 221
Height and width of the drawing pane 256
Setting the Height and Width of the FED Drawing Pane 256
Character *Height* Font Attribute 221
Blinker Width and *Blinker* *Height* Font Attributes 223
Character Height of the FED Character Box 238
HELP 44
Automatic drawing help 243
Dired Help 148
Finding Out About Zmacs Commands with HELP 44
Introduction to HELP 14
Introduction to Zmacs Help 14
Minibuffer Response Help 51
Zmacs Help 14
HELP ? Zmacs command 14
Help a 45
Example of a Search String for HELP A 46
HELP A Zmacs command 14, 45
Help C 44
More HELP Commands for Finding Out About Zmacs Commands 46
Zmacs Help Command Summary 217
HELP completion command 14
HELP C Zmacs command 14, 45
Help D 45
HELP Dired command 148
HELP D Zmacs command 14, 45
HELP Font Editor command 260
[HELP] Font Editor menu item 260
Getting Help in Zmacs 41
HELP key 14, 44

Help L 46
HELP L Zmacs command 14, 46
HELP or c-? 14
HELP SPACE Zmacs command 14
Help U 46
HELP U Zmacs command 14, 46
Help V 46
HELP V Zmacs command 14, 46, 210
Help W 46
HELP W Zmacs command 14
HELP W Zmacs command 46
Marking a Region From Here to Beginning of Buffer 92
Marking a Region From Here to End of Buffer 92
H Font Editor command 259
H Font Editor command 255
What Histories Save 72
Buffer History 116
Command history 43
Kill history 43, 73, 75
Viewing the Editor Command History 73
Viewing the Kill History 73
Yanking in the command history 15
Yanking in the kill history 15
Zmacs Buffer History 116
Retrieving History Elements 74
Using the Mouse on History Elements 74
History list 15
Setting Mode Hooks in Init Files 214
Moving the Drawing Horizontally and/or Vertically in FED 255
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Contracting a Drawing Horizontally in FED 250
Stretching a Drawing Horizontally in FED 250
Delete Horizontal Space 28
How Key Bindings Work: the Comtab 208
How Tag Tables Work 106
How They Work 137
How to Create an Environment 33
How to Specify Zmacs Variable Settings 210
How to Use Formatting Commands 36
How to Use the **loop** Indentor 163
How Zmacs Keyboard Macros Work 199
How Zmacs Minor Modes Work 195

@i 34
Fast Where Am I 47
SELECT I 40
Where Am I 47
Ibase global variable 141, 143
Mirror imaging characters 243
Inactive patches 190
Zmacs Incremental Search 98
Zmacs Reverse Incremental Search 99
Delete Indentation 28
Zwei:*inhibit-fancy-loop indentation 162
Indentation in **loop** Macros 162
Indentation in Zmacs 162

Aligning Indentation in Zmacs 165
Changing Case and Indentation in Zmacs 159
Deleting Indentation in Zmacs 166
New Line with This Indentation in Zmacs 166
Overview of Indentation in Zmacs 162
Going Back to First Indented Character in Zmacs 165
Creating a New Indented Lisp Comment Line in Zmacs 172
Indenting Current Line in Zmacs 162
Indenting for Lisp Comment in Zmacs 171
Indenting New Line in Zmacs 164
Indenting Region in Zmacs 165
Indenting Region Uniformly in Zmacs 165
How to Use the **loop** Indentor 163
The **loop** Indentor 162
Loop Indentor Example 1 163
Loop Indentor Example 2 163
Exclamation point (!) line continuation indicator 23, 54
Semicolon (;) comment indicator 171
General Information-giving Zmacs Commands 46
Zwei: *inhibit-fancy-loop indentation 162
Creating an Init File 213
Init File Form: Auto Fill in Text Mode 214
Init File Form: Balanced Quotation Marks and Asterisks 215
Init File Form: c-m-L on the SQUARE Key 215
Init File Form: Edit Buffers on c-X c-B 215
Init File Form: Edit Buffers on m-X 216
Init File Form: Electric Shift Lock in Lisp Mode 214
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: m-. on m-(L) 216
Init File Form: Ordering Buffer Lists 213
Init File Form: Putting Buffers Into Current Package 213
Init File Form: Setting Default Major Mode 213
Init File Form: Setting Find File Not to Create New Files 214
Init File Form: Setting Goal Column for Real Line Commands 214
Init File Form: White Space in Lisp Code 215
Init Files 213
Customizing Zmacs in Init Files 213
Introduction to Customizing Zmacs in Init Files 213
Setting Editor Variables in Init Files 213
Setting Key Bindings in Init Files 215
Setting Mode Hooks in Init Files 214
Initial patch state 190
In-progress patch 190
In-progress patch state 190
Mouse as a graphic input device 227
Insert Buffer (m-X) Zmacs command 124
Insert File (m-X) Zmacs command 124
Inserting a Buffer Into Another Buffer 124
Inserting a File Into a Buffer 124
Inserting and Removing Lisp Comments From Regions in Zmacs 173
Inserting Blank Line in Zmacs 167
Inserting Characters 22

Inserting Formatting Characters 23
Saving and Inserting Regions in Registers 89
Inserting Text 22
Introduction to Inserting Text 22
Appending, Prepending, and Inserting Text in Zmacs 124
Insert Matching parentheses 178
Insert text from register into buffer 90
Entering Inspector 40
Install Command 209
Install Command (m-X) Zmacs command 209
Source Compare Installed Definition 127
Source Compare Merge Installed Definition 128
Installing a Macro on a Key 203
Installing a Mouse Macro 203
Example of Installing and Deinstalling a Macro 204
Install Macro (m-X) Zmacs command 203
Install patch file 190
Zmacs Commands for Interacting with Lisp 218
Defining an Interactive Keyboard Macro 202
FED Outside World Interface Menu Items 260
Introduction to Completion 14
Introduction to Customizing Zmacs 194
Introduction to Customizing Zmacs in Init Files 213
Introduction to Deleting and Transposing Lines 81
Introduction to Deleting and Transposing Lisp Expressions 79
Introduction to Deleting and Transposing Words 78
Introduction to Deleting Sentences 83
Introduction to Editing Lisp Programs in Zmacs 170
Introduction to Entering Zmacs 12
Introduction to Erasing Text 28
Introduction to Fonts 221
Introduction to HELP 14
Introduction to Inserting Text 22
Introduction to Locating Source Code in Zmacs 180
Introduction to Motion by Page 69
Introduction to Motion by Paragraph 68
Introduction to Moving the Cursor 26
Introduction to Redisplaying the Window 55
Introduction to Regions 86
Introduction to Tag Tables and Search Domains 106
Introduction to Text Formatting in Zmacs 33
Introduction to the Motion Commands 60
Introduction to the Organization of the Screen 17
Introduction to the Zmacs Manual 3
Introduction to Using the Mouse 57
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 6
Invoking Zmacs 12
What is a Zmacs Region? 86
Italics 34
Itemize text 34
FED Gray Plane Menu Items 260

FED Outside World Interface Menu Items 260

K **K** **K**

Start Kbd Macro 199
Name Last Kbd Macro (m-X) Zmacs command 203
View Kbd Macro (m-X) Zmacs command 200
Kbd Macro Query 202
End Kbd Macro Zmacs command 199
K Dired command 151
Left kern 222
Left *Kern* Font Attribute 222
HELP key 14, 44
Init File Form: c-m-L on the SQUARE Key 215
Installing a Macro on a Key 203
Leaving Zmacs with the SELECT Key 40
RUBOUT key 22
SELECT key 40
Setting the Key 208
Assign key bindings 208
Definition of Key Bindings 208
Extended command key bindings 46
Zmacs Key Bindings 208
Setting Key Bindings in Init Files 215
How Key Bindings Work: the Comtab 208
FED Menu and Keyboard Commands 259
Calling the Last Keyboard Macro 200
Defining an Interactive Keyboard Macro 202
Definition of a Zmacs Keyboard Macro 199
Ending a Keyboard Macro 200
Example of Calling the Last Keyboard Macro 200
Naming a Keyboard Macro 203
Starting a Keyboard Macro 200
Viewing a Keyboard Macro 200
Creating New Zmacs Commands with Keyboard Macros 199
Example 1 of Making Tables Using Keyboard Macros 206
Example 1 of Writing and Saving Keyboard Macros 201
Example 2 of Making Tables Using Keyboard Macros 206
Example 2 of Writing and Saving Keyboard Macros 202
Making Tables Using Keyboard Macros 205
Procedure for Creating Zmacs Commands with Keyboard Macros 199
Sort Via Keyboard Macros 203
Writing and Saving Keyboard Macros 201
Using Keyboard Macros to Sort 203
How Zmacs Keyboard Macros Work 199
FED Keyboard-only Commands 261
RETURN key in the minibuffer 51
Shift keys 6
Getting Out of Keystroke Prefixes 42
Keystrokes 6
Creating a Region with Keystrokes 87
Displaying previous keystrokes 46
Introduction to Zmacs Keystrokes 6
List the last sixty keystrokes 46
Using the CONTROL key while drawing characters 235
Using the META key while drawing characters 235
:default-font keyword to **si:*hardcopy-default-fonts*** 271
:header-font keyword to **si:*hardcopy-default-fonts*** 271

Append Next Kill 75
Query Replace Last Kill 104
Init File Form: Fixing White Space for Kill/Yank Commands 214
Kill Backward Up List (c-m-X) Zmacs command 79
Kill commands 28
Kill history 43, 73, 75
Viewing the Kill History 73
Yanking in the kill history 15
Killing a Lisp Comment in Zmacs 171
Deleting Vs. Killing Text 72
Overview of Deleting Vs. Killing Text 72
Kill Line 28, 81
Kill Merging 75
Kill Region 28
Kill Sentence 28, 83
Backward Kill Sentence 28
Kill Sexp 28, 79
Backward Kill Sexp 28, 79
Kill Word 28, 78
Backward Kill Word 28, 78

L

L

L

Help L 46
SELECT L 40
L2:Move to point mouse click 57
Large Deletions 43
Deleting the Last Character 76
Cancel last command 42
Name Last Kbd Macro (m-X) Zmacs command 203
Calling the Last Keyboard Macro 200
Example of Calling the Last Keyboard Macro 200
Query Replace Last Kill 104
Repeat Last Minibuffer Command 51, 74
List the last sixty commands 46
List the last sixty keystrokes 46
Entering and Leaving FED 225
Leaving Zmacs 40
Overview of Leaving Zmacs 40
Leaving Zmacs Via the System Menu 40
Leaving Zmacs with c-Z 40
Leaving Zmacs with the SELECT Key 40
Left and Right Edges of the FED Character Box 237
Left kern 222
Left Kern Font Attribute 222
Query Replace Let Binding 104
Abort At Top Level 42
Motion Along One Nesting Level 65
Motion up and Down Nesting Levels 66
Beginning of Line 26, 63, 81
Breaking a line 22
Deleting Backward on the Line 82
Deleting the Current Line 81
Down Line 63, 81
Down Real Line 26, 63, 81
Draw a line 259
End of Line 26, 63, 81
Erase backward to start of line 82

Kill Line 28, 81
Motion by Line 63
Mouse Documentation Line 57
Move cursor to beginning of line 56
Moving to a Specified Line 56
Starting a New Line 22
Up Line 63, 81
Up Real Line 26, 63, 81
Zmacs Mode Line 19
Mode Line's *Buffer* 20
Mode Line's *Buffer-status* 21
Mode Line's *Major-mode* 20
Mode Line's *Minor-mode* 20
Mode Line's *Position-flag* 21
Mode Line's *Version* 21
Init File Form: Setting Goal Column for Real Line Commands 214
Exclamation point (!) line continuation indicator 23, 54
Status line documentation 255
Moving Rest of Line Down in Zmacs 166
Mode Line Example 21
[Draw Line] Font Editor menu item 246, 259
Line height 221
Centering the Current Line in Zmacs 164
Creating a New Indented Lisp Comment Line in Zmacs 172
Deleting Blank Line in Zmacs 167
Indenting Current Line in Zmacs 162
Indenting New Line in Zmacs 164
Inserting Blank Line in Zmacs 167
Mouse Documentation Line in Zmacs 57
Moving Down to Lisp Comment on Next Line in Zmacs 172
Moving up to Lisp Comment on Previous Line in Zmacs 172
Delete Blank Lines 28
Deleting and Transposing Lines 81
Exchange Lines 82
Introduction to Deleting and Transposing Lines 81
Merging lines 22
Wraparound Lines 54
Wrapping Lines 23
Drawing Lines and Curves in FED 246
Wraparound Lines in the Editor Window 54
What the Lines Mean in the FED Character Box 237
Transposing Lines of Text 82
Count Lines Page 48
Count Lines Region 48
New Line with This Indentation in Zmacs 166
LINE Zmacs command 164
Create Link 135
Create Link (m-X) Zmacs command 135
Creating Links to Files 135
Base and Syntax Default Settings for Lisp 31, 120, 156, 170
Entering Lisp 40
Setting the Syntax for Symbolics Common Lisp 140
Zmacs Commands for Finding Out About Lisp 217
Zmacs Commands for Interacting with Lisp 218
Init File Form: White Space in Lisp Code 215
Commenting Lisp Code in Zmacs 171
Overview of Commenting Lisp Code in Zmacs 171
Setting the Lisp Comment Column in Zmacs 172

Indenting for Lisp Comment in Zmacs 171
Killing a Lisp Comment in Zmacs 171
Creating a New Indented Lisp Comment Line in Zmacs 172
Moving Down to Lisp Comment on Next Line in Zmacs 172
Moving up to Lisp Comment on Previous Line in Zmacs 172
Inserting and Removing Lisp Comments From Regions in Zmacs 173
Lisp Compiler Warnings 176
Deleting the Current Lisp Expression 79
Deleting the List Containing the Current Lisp Expression 79
Deleting the Previous Lisp Expression 79
Description of Motion by Lisp Expression 65, 79
Motion by Lisp Expression 65
Commands to Mark Regions by Lisp Expressions 91
Deleting and Transposing Lisp Expressions 79
Introduction to Deleting and Transposing Lisp Expressions 79
Transposing Lisp Expressions 79
Expanding Lisp Expressions in Zmacs 179
Parenthesizing Lisp Expressions in Zmacs 178
(fed) Lisp Listener command 225
Init File Form: Electric Shift Lock in Zmacs Lisp Mode 214
Lisp Mode 156
Lisp Mode (m-X) Zmacs command 170
Compiling Lisp Programs in Zmacs 175
Editing Lisp Programs in Zmacs 169
Evaluating Lisp Programs in Zmacs 174
Evaluating and Compiling Lisp Programs in Zmacs 174
Introduction to Editing Lisp Programs in Zmacs 170
Overview of Evaluating and Compiling Lisp Programs in Zmacs 174
fs:file-type-mode-alist Lisp variable 198
zwei:*major-mode-translations* Lisp variable 198
Attribute list 137
Backward List 65
Backward up List 66
Describe Attribute List 149
Display argument list 47, 48
Down List 66
FED Command List 259
Forward List 65
Forward up List 66
History list 15
List Buffers 118
List Buffers Zmacs command 115
List Changed Definitions of Buffer (m-X) Zmacs command 183
Kill Backward Up List (c-m-X) Zmacs command 79
Deleting the List Containing the Current Lisp Expression 79
List contents of a directory 131
The Zmacs List Definition Commands 182
List Definitions (m-X) Zmacs command 183
(fed) Lisp Listener command 225
Using the mouse with List Files 131
List Files (m-X) Zmacs command 131
Mousing on the FED List Fonts and Show Font Displays 263
[List Fonts] Font Editor menu item 230, 260
Using the mouse with [List Fonts] Font Editor menu item 263
List Fonts (m-X) Zmacs command 223, 271
Selecting, Listing, and Examining Zmacs Buffers 116
Listing Buffers 117

Example of Listing Buffers 118
Listing Files in a Directory 131
Specifying and Listing Tag Tables 106
Listing Variables by Matching a String 210
Example of Listing Variables by Matching a String 211
Listing Zmacs Variables 210
Reparse Attribute List (m-X) Zmacs command 137
Update Attribute List (m-X) Zmacs command 138
Update Attribute List Query 141
Attribute lists 138
Init File Form: Ordering Buffer Lists 213
Warnings about file attribute lists 138
List the last sixty commands 46
List the last sixty keystrokes 46
List Variables 210
List Variables (m-X) Zmacs command 210
L:Move point mouse click 57
Loading a File in Dired 148
Overview of Locating and Replacing Strings Automatically 102
Locating and Replacing Strings Automatically in Zmacs 102
Introduction to Locating Source Code in Zmacs 180
Locating Source Code to Edit in Zmacs 180
Saving and Moving to Locations in Registers 89
Init File Form: Electric Shift Lock in Lisp Mode 214
Long Documentation 47
How to Use the **loop** Indentor 163
The **loop** Indentor 162
Loop Indentor Example 1 163
Loop Indentor Example 2 163
TAB in **loop** macro 162
Indentation in **loop** Macros 162
Lowercase Attribute 142
Lowercase file attribute 142, 143
Set Lowercase (m-X) Zmacs command 142, 143
c-X L Zmacs command 48
HELP L Zmacs command 14, 46

M M M

SELECT M 40
Example of the m-. Command 181
Init File Form: m-. on m-(L) 216
M2:Save/Kill/Yank mouse click 57
m-; Zmacs command 171
m-< Zmacs command 26, 70
m-= Zmacs command 48
m-> Zmacs command 26, 70
Calling the Last Keyboard Macro 200
Defining an Interactive Keyboard Macro 202
Definition of a Zmacs Keyboard Macro 199
Deinstalling a Macro 204
Ending a Keyboard Macro 200
Example of Calling the Last Keyboard Macro 200
Example of Installing and Deinstalling a Macro 204
Installing a Mouse Macro 203
Naming a Keyboard Macro 203
Starting a Keyboard Macro 200

Start Kbd Macro 199
TAB in **loop** macro 162
Viewing a Keyboard Macro 200
Deinstall Macro (m-X) Zmacs command 204
Install Macro (m-X) Zmacs command 203
Name Last Kbd Macro (m-X) Zmacs command 203
View Kbd Macro (m-X) Zmacs command 200
Installing a Macro on a Key 203
Kbd Macro Query 202
Creating New Zmacs Commands with Keyboard Macros 199
Example 1 of Making Tables Using Keyboard Macros 206
Example 1 of Writing and Saving Keyboard Macros 201
Example 2 of Making Tables Using Keyboard Macros 206
Example 2 of Writing and Saving Keyboard Macros 202
Indentation in **loop** Macros 162
Making Tables Using Keyboard Macros 205
Procedure for Creating Zmacs Commands with Keyboard Macros 199
Sort Via Keyboard Macros 203
Writing and Saving Keyboard Macros 201
Using Keyboard Macros to Sort 203
How Zmacs Keyboard Macros Work 199
End Kbd Macro Zmacs command 199
Zmacs Macsyma Mode 157
Send mail about patch 190
Zmacs Major Editing Modes 156
Default major mode 198
Init File Form: Setting Default Major Mode 213
Overview of Setting the Major Mode 156
Setting the Zmacs Major Mode 155
Mode Line's *Major-mode* 20
File Types and Zmacs Major Modes 198
User-defined Zmacs Major Modes 198
Zmacs Major Modes 198
zwei: ***major-mode-translations*** Lisp variable 198
:convert-to-device-units method of **si:** **make-hardcopy-stream** 275
:eject-page method of **si:** **make-hardcopy-stream** 274
:set-cursorpos method of **si:** **make-hardcopy-stream** 274
:set-font method of **si:** **make-hardcopy-stream** 274
:show-line method of **si:** **make-hardcopy-stream** 275
:show-rectangle method of **si:** **make-hardcopy-stream** 274
:un-relative-coordinates method of **si:** **make-hardcopy-stream** 275
si: **make-hardcopy-stream** function 274
zwei: **make-macro-command** 201
Make region 57
Making Global Replacements in Zmacs 102
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
Making Patches 186
Making Tables Using Keyboard Macros 205
Example 1 of Making Tables Using Keyboard Macros 206
Example 2 of Making Tables Using Keyboard Macros 206
Manipulating Buffers and Files in Zmacs 113
Retrieving the Black Plane While Manipulating FED Registers 241
Commands for manipulating files 131
Overview of Zmacs File Manipulation Commands 131
Zmacs File Manipulation Commands 131
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
[Map over] Zmail menu item 268
Region Right Margin Mode Variable 211
Set Pop Mark 88
Setting/Popping the Mark 88
Showing the Mark 88
Swap Point And Mark 88
Mark and the Region 86
Mark Beginning 92
Mark Definition 91
Mark End 92
Marking a Region From Here to Beginning of Buffer 92
Marking a Region From Here to End of Buffer 92
Marking Files for Deletion in Dired 150
Marking Files to Be Hardcopied in Dired 153
Region Marking Mode Variable 211
Marking text 91
Mark Page 92
Mark Paragraph 92
Mark region 57
Commands to Mark Regions 91
Overview of Commands to Mark Regions 91
Commands to Mark Regions by Buffers 92
Commands to Mark Regions by Lisp Expressions 91
Commands to Mark Regions by Pages 92
Commands to Mark Regions by Paragraphs 92
Example of Commands to Mark Regions by Paragraphs 92
Commands to Mark Regions by Words 91
Init File Form: Balanced Quotation Marks and Asterisks 215
Mark Sexp 91
M: Mark thing mouse click 57
Example of Listing Variables by Matching a String 211
Listing Variables by Matching a String 210
Insert Matching parentheses 178
m-A Zmacs command 26
m-B Zmacs command 26, 61
m-C Zmacs command 160
m-D Zmacs command 28, 78
Current meaning of mouse clicks 57
What the Lines Mean in the FED Character Box 237
Character Select menu 229
Dired Menu 148, 149
Drawing Pane Menu 228
Draw Mode Menu 228, 235, 262
FED Character Select Menu 229
FED Font Parameters Menu 229
Font Parameters menu 229
Gray Plane Menu 228
Hardcopying From the System Menu 267
Leaving Zmacs Via the System Menu 40
Mousing on the FED Draw Mode Menu 262
Mousing on the FED Font Parameters Menu 263
Outside FED Command Menu 229

Overview of the Editor Menu 49
Selecting a FED Character From the Character Select Menu 233
The Editor Menu 49
Using the mouse in the draw mode menu 262
Using the mouse in the Font Parameters menu 263
FED Menu and Keyboard Commands 259
Editor Menu Commands 49
[Add In Gray] Font Editor menu item 240, 260
[Center View] Font Editor menu item 255, 259
[Clear Gray] Font Editor menu item 239, 260
[Clear Points] Font Editor draw mode menu item 235
[Configure] Font Editor menu item 227, 256, 259
[Draw Line] Font Editor menu item 246, 259
[Draw Spline] Font Editor menu item 246, 259
[Edit Font] Font Editor menu item 230, 260
[Erase All] Font Editor menu item 239, 243, 259
[EXIT] Font Editor menu item 260
[Flip Points] Font Editor draw mode menu item 235
[Gray Char] Font Editor menu item 239, 260
[Grid Size] Font Editor menu item 256, 259
[Hardcopy] Zmail menu item 268
[HELP] Font Editor menu item 260
[List Fonts] Font Editor menu item 230, 260
[Map over] Zmail menu item 268
[Move Black] Font Editor menu item 238, 259
[Move Gray] Font Editor menu item 239, 260
[Move View] Font Editor menu item 255, 259
[Move] Zmail menu item 268
[Read File] Font Editor menu item 258, 260
[Reflect] Font Editor menu item 243, 259
[Rename Char] Font Editor menu item 234, 260
[Rotate] Font Editor menu item 243, 259
[Save Char] Font Editor menu item 234, 260
[Set Points] Font Editor draw mode menu item 235
[Set Sample] Font Editor menu item 253, 260
[Show Font] Font Editor menu item 227, 233, 234, 260
[Stretch] Font Editor menu item 259
[Swap Gray] Font Editor menu item 239, 260
Using the mouse with [List Fonts] Font Editor menu item 263
Using the mouse with [Show Font] Font Editor menu item 234, 263
[Write File] Font Editor menu item 258, 260
FED Gray Plane Menu Items 260
FED Outside World Interface Menu Items 260
R: Menu mouse click 57
R2:System menu mouse click 57
Command menus 228
FED Menus 228
Using the mouse with Font Editor menus 228
Source Compare Merge 125
Source Compare Merge Installed Definition 128
Source Compare Merge (m-X) Zmacs command 125
Source Compare Merge Newest Definition 127
Source Compare Merge Saved Definition 127
Kill Merging 75
Merging Characters with the FED Gray Plane 240
Merging lines 22
m-ESCAPE Zmacs command 175

Printing messages 268
Using the META key while drawing characters 235
Method for Searching for Appropriate Zmacs Commands 45
:convert-to-device-units method of **si:make-hardcopy-stream** 275
:eject-page method of **si:make-hardcopy-stream** 274
:set-cursorpos method of **si:make-hardcopy-stream** 274
:set-font method of **si:make-hardcopy-stream** 274
:show-line method of **si:make-hardcopy-stream** 275
:show-rectangle method of **si:make-hardcopy-stream** 274
:un-relative-coordinates method of **si:make-hardcopy-stream** 275
m-E Zmacs command 62
M Font Editor command 260
m-F Zmacs command 26, 61
m-H Zmacs command 92
Zmacs Midas Mode 157
Minibuffer 19
Echo Area's Minibuffer 19
Evaluate Minibuffer 175
More on the Minibuffer 51
RETURN key in the minibuffer 51
Yanking in the Minibuffer 51
Repeat Last Minibuffer Command 51, 74
Minibuffer Prompts 42
Getting Out of Minibuffer Prompts 42
Minibuffer Response Format 51
Minibuffer Response Help 51
More Ways to Enter Minibuffer Responses 51
Example of Filling Text with Auto Fill Minor Mode 195
Mode Line's *Minor-mode* 20
Built-in Customization Using Zmacs Minor Modes 195
Definition of Zmacs Minor Modes 195
Summary of Zmacs Minor Modes 196
How Zmacs Minor Modes Work 195
Minor version number 186
Mirror imaging characters 243
m-K Zmacs command 28, 83
Init File Form: m-. on m-(L) 216
m-LINE Zmacs command 172
m-L Zmacs command 160
M:Mark thing mouse click 57
m-N Zmacs command 172
Auto Fill Mode 142, 143, 196
Default major mode 198
Example of Filling Text with Auto Fill Minor Mode 195
Init File Form: Auto Fill in Text Mode 214
Init File Form: Electric Shift Lock in Lisp Mode 214
Init File Form: Setting Default Major Mode 213
Overview of Setting the Major Mode 156
Setting the Zmacs Major Mode 155
Two window mode 129
Zmacs Bolio Mode 157
Zmacs Electric PI1 Mode 157
Zmacs Fortran Mode 156
Zmacs Fundamental Mode 156
Zmacs Lisp Mode 156
Zmacs Macsyma Mode 157
Zmacs Midas Mode 157

Zmacs Pli Mode 157
Zmacs Teco Mode 157
Zmacs Text Mode 156
Creating a Fundamental Mode Buffer 122
Setting Mode Hooks in Init Files 214
Dired Mode in Zmacs 144
Zmacs Mode Line 19
Mode Line's *Buffer* 20
Mode Line's *Buffer-status* 21
Mode Line's *Major-mode* 20
Mode Line's *Minor-mode* 20
Mode Line's *Position-flag* 21
Mode Line's *Version* 21
Mode Line Example 21
Draw Mode Menu 228, 235, 262
Mousing on the FED Draw Mode Menu 262
Using the mouse in the draw mode menu 262
[Clear Points] Font Editor draw mode menu item 235
[Flip Points] Font Editor draw mode menu item 235
[Set Points] Font Editor draw mode menu item 235
Find File in Fundamental Mode (m-X) Zmacs command 122
Lisp Mode (m-X) Zmacs command 170
Built-in Customization Using Zmacs Minor Modes 195
Definition of Zmacs Minor Modes 195
File Types and Zmacs Major Modes 198
Summary of Zmacs Minor Modes 196
User-defined Zmacs Major Modes 198
Zmacs Major Modes 198
Zmacs Major Editing Modes 156
How Zmacs Minor Modes Work 195
Region Marking Mode Variable 211
Region Right Margin Mode Variable 211
Modification flag 114
Modified Two Windows 129
More HELP Commands for Finding Out About Zmacs Commands 46
More on the Minibuffer 51
More Ways to Enter Minibuffer Responses 51
Motion 57
Motion Along One Nesting Level 65
Motion Among Top-level Expressions 66
Motion by Character 60
Motion by Line 63
Motion by Lisp Expression 65
Description of Motion by Lisp Expression 65, 79
Motion by Page 69
Introduction to Motion by Page 69
Motion by Paragraph 68
Introduction to Motion by Paragraph 68
Motion by Sentence 62
Motion by Word 61
Motion Commands 60
Example of Negative Numeric Arguments with Motion Commands 60
Example of Numeric Arguments with Motion Commands 60
Goal Column and the Motion Commands 63
Introduction to the Motion Commands 60
Negative Numeric Arguments and Motion Commands 60
Numeric Arguments and the Motion Commands 60

Motion up and Down Nesting Levels 66
Motion with Respect to the Whole Buffer 70
Mouse 12, 57, 118, 125
Creating a Region with the Mouse 87
Drawing Characters in FED with the Mouse 235
Drawing characters with the mouse 235
Entering Zmacs with the **Mouse** 12
Introduction to Using the Mouse 57
Moving the Cursor with the Mouse 57
Mouse as a graphic input device 227
L2:Move to point mouse click 57
L:Move point mouse click 57
M2:Save/Kill/Yank mouse click 57
M:Mark thing mouse click 57
R2:System menu mouse click 57
R:Menu mouse click 57
Current meaning of mouse clicks 57
Mouse Documentation Line 57
Mouse Documentation Line in Zmacs 57
Using the mouse in the character select pane 263
Using the mouse in the drawing pane 262
Using the mouse in the draw mode menu 262
Using the mouse in the Font Parameters menu 263
Using the mouse in the register pane 263
Using the mouse in the sample pane 262
Installing a Mouse Macro 203
Using the Mouse on History Elements 74
Using the mouse on the character box 238
Mouse-sensitive Zmacs commands 57
Mouse Sensitivities in FED 262
Using the mouse to enter Zmacs 12
Using the mouse with Font Editor menus 228
Using the mouse with List Files 131
Using the mouse with [List Fonts] Font Editor menu item 263
Using the mouse with [Show Font] Font Editor menu item 234, 263
Mousing on the FED Character Select Pane 263
Mousing on the FED Drawing Pane 262
Mousing on the FED Draw Mode Menu 262
Mousing on the FED Font Parameters Menu 263
Mousing on the FED List Fonts and Show Font Displays 263
Mousing on the FED Register Pane 263
Mousing on the FED Sample Pane 262
[Move Black] Font Editor menu item 238, 259
Move cursor to beginning of line 56
Move drawing in the gray plane 239, 260
[Move Gray] Font Editor menu item 239, 260
Summary of Cursor Movement 54
Cursor movement commands 26, 60
Move Over) 67
Dired move point 149
L: Move point mouse click 57
Move to Default Previous Point 88
L2: Move to point mouse click 57
Move to Previous Point 88
[Move View] Font Editor menu item 255, 259
[Move] Zmail menu item 268

Moving Around in Dired 149
Moving Down to Lisp Comment on Next Line in Zmacs 172
Moving Rest of Line Down in Zmacs 166
Moving the cursor 57
Description of Moving the Cursor 26
Introduction to Moving the Cursor 26
Overview of Moving the Cursor 54
Summary of Moving the Cursor 26
Moving the Cursor in Zmacs 53
Moving the Cursor with the Mouse 57
Moving the drawing 246, 259
Moving the Drawing Horizontally and/or Vertically in FED 255
Moving the Drawing in FED 246
Moving to a Specified Line 56
Moving to end of buffer 70
Saving and Moving to Locations in Registers 89
Moving to Previous Points 88
Moving up to Lisp Comment on Previous Line in Zmacs 172
m-O Zmacs command 166
m-P Zmacs command 172
FUNCTION m-Q 268
m-RUBOUT Zmacs command 28, 78
m-R Zmacs command 56
m-SCROLL Zmacs command 26, 56
m-sh-D Zmacs command 47
m-sh-E Zmacs command 175
m-S Zmacs command 164
m-T Zmacs command 78
Multiple buffers 114
Deleting Multiple File Versions in Dired 151
Hardcopy files with multiple fonts 273
Querying While Making Multiple Global Replacements in Zmacs 103
Deleting Multiple Versions 135
Multiple windows 114
m-U Zmacs command 160
m-V Zmacs command 26, 56
m-W Zmacs command 94
m-X 7
Init File Form: Edit Buffers on m-X 216
Abort Patch (m-X) 191
Add Patch (m-X) 188
Add Patch Changed Definitions (m-X) 189
Add Patch Changed Definitions of Buffer (m-X) 188
Arglist (M-x) 48
Finish Patch (m-X) 190
Recompile Patch (m-X) 192
Reload Patch (m-X) 192
Resume Patch (m-X) 191
Select Patch (m-X) 190
Set Syntax [m-X) 140
Start Patch (m-X) 187
Start Private Patch (m-X) 188
View Patches (m-X) 190
Append To File (m-X) Zmacs command 124
Arglist (m-X) Zmacs command 48

Change File Properties (m-X) Zmacs command 133
Clean Directory (m-X) Zmacs command 136
Compile Region (m-X) Zmacs command 94
Copy File (m-X) Zmacs command 134
Create Directory (m-X) Zmacs command 131
Create Link (m-X) Zmacs command 135
Deinstall Macro (m-X) Zmacs command 204
Delete File (m-X) Zmacs command 135
Describe Variable (m-X) Zmacs command 210
Dired (m-X) Zmacs command 144
Edit Buffers (m-X) Zmacs command 118
Edit Directory (m-X) Zmacs command 144
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Evaluate Buffer (m-X) Zmacs command 175
Evaluate Changed Definitions (m-X) Zmacs command 175
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
Evaluate Into Buffer (m-X) Zmacs command 175
Evaluate Region (m-X) Zmacs command 175
Find File In Fundamental Mode (m-X) Zmacs command 122
Find Unbalanced Parentheses (m-X) Zmacs command 48
Format Buffer (m-X) Zmacs command 33, 38
Format File (m-X) Zmacs command 33, 38
Format Region (m-X) Zmacs command 33, 38
Hardcopy Buffer (m-X) Zmacs command 119, 267
Insert Buffer (m-X) Zmacs command 124
Insert File (m-X) Zmacs command 124
Install Command (m-X) Zmacs command 209
Install Macro (m-X) Zmacs command 203
Lisp Mode (m-X) Zmacs command 170
List Changed Definitions of Buffer (m-X) Zmacs command 183
List Definitions (m-X) Zmacs command 183
List Files (m-X) Zmacs command 131
List Fonts (m-X) Zmacs command 223, 271
List Variables (m-X) Zmacs command 210
Name Last Kbd Macro (m-X) Zmacs command 203
Prepend To File (m-X) Zmacs command 124
Query Replace (m-X) Zmacs command 102
Reap File (m-X) Zmacs command 136
Rename Buffer (m-X) Zmacs command 119
Rename File (m-X) Zmacs command 133
Reparse Attribute List (m-X) Zmacs command 137
Replace String (m-X) Zmacs command 102
Revert Buffer (m-X) Zmacs command 121
Set Backspace (m-X) Zmacs command 141, 143
Set Base (m-X) Zmacs command 141, 143
Set Fonts (m-X) Zmacs command 141, 143
Set Lowercase (m-X) Zmacs command 142, 143
Set Nofill (m-X) Zmacs command 142, 143
Set Package (m-X) Zmacs command 138, 143
Set Patch File (m-X) Zmacs command 143
Set Tab Width (m-X) Zmacs command 143
Set Variable (m-X) Zmacs command 212
Set Visited File Name (m-X) Zmacs command 122
Set Vsp (m-X) Zmacs command 143
Show Directory (m-X) Zmacs command 132
Show File (m-X) Zmacs command 132
Source Compare (m-X) Zmacs command 125
Source Compare Merge (m-X) Zmacs command 125

Trace (m-X) Zmacs command 48
Uncomment Region (m-X) Zmacs command 173
Update Attribute List (m-X) Zmacs command 138
View Buffer (m-X) Zmacs command 119
View Directory (m-X) Zmacs command 132
View File (m-X) Zmacs command 132
View File Properties (m-X) Zmacs command 133
View Kbd Macro (m-X) Zmacs command 200
m-Y yank command 15, 51, 74
m-% Zmacs command 102
m-) Zmacs command 67
m-[Zmacs command 26, 68
m-] Zmacs command 26, 68
m-@ Zmacs command 91
m-Z Zmacs command 176
m-\ Zmacs command 28
m-^ Zmacs command 28, 166

N N N

SELECT N 40
Name Last Kbd Macro (m-X) Zmacs command 203
Set Visited File Name (m-X) Zmacs command 122
Command Names 6
Zmacs Buffer and File Names 114
Names of commands 7, 45
Naming a File 121
Naming a Keyboard Macro 203
Negative Numeric Arguments and Motion Commands 60
Example of Negative Numeric Arguments with Motion Commands 60
Motion Along One Nesting Level 65
Motion up and Down Nesting Levels 66
Reading a File Into a New Buffer 120
Selecting a FED Character by Creating a New Character 233
Creating new characters 234
Source Compare Newest Definition 127
Source Compare Merge Newest Definition 127
Buffer Flags for New Files 115
Init File Form: Setting Find File Not to Create New Files 214
Creating a New Font in FED 230
Creating new fonts 230
Creating a New Indented Lisp Comment Line in Zmacs 172
Starting a New Line 22
Newline characters 22
Indenting New Line in Zmacs 164
Newlines 22
New Line with This Indentation in Zmacs 166
Creating New Zmacs Commands with Keyboard Macros 199
Append Next Kill 75
Moving Down to Lisp Comment on Next Line in Zmacs 172
Next Page 26, 69
Next Possibility 110
Displaying the Next Possibility 110
Example of Displaying the Next Possibility 110
Next Screen 26
Displaying the Next Screen 55

Dired Next Undumped 153
Nofill Attribute 142
Nofill file attribute 142, 143
Set Nofill (m-X) Zmacs command 142, 143
Nonmouse cursor 235, 236, 261
The FED Nonmouse Cursor 236
Dired Complement No Reap Flag 152
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 152
@note 37
note-private-patch function 192
Entering Notifications 40
Init File Form: Setting Find File Not to Create New Files 214
Nullifying prefixes 42
Minor version number 186
Quadruple Numeric Arg 24
Numeric arguments 22, 24, 45
Defaults to Numeric Arguments 24
Example of Numeric Arguments 24
Overview of Numeric Arguments 24
Negative Numeric Arguments and Motion Commands 60
Numeric Arguments and the Motion Commands 60
Example of Numeric Arguments with Motion Commands 60
Example of Negative Numeric Arguments with Motion Commands 60

O O O

Motion Along One Nesting Level 65
One Window 130
Returning to One Window 130
One Window Default Variable 211
Online documentation for commands 45
Online Documentation for Dired 148
Online documentation for prefixes 45
Example of a Tag Tables Replacement Operation 106
Other Types of Replacement Operations in Zmacs 104
Performing Operations with Tag Tables 107
Init File Form: Ordering Buffer Lists 213
Organization of the Screen 17
Introduction to the Organization of the Screen 17
Organization of the Zmacs Manual 4
Creating Two Windows, Specifying Other Contents 129
Other Hardcopy Commands 269
Other Region-related Commands 95
Other Set Commands for File and Buffer Attributes 140
Other Types of Replacement Operations in Zmacs 104
Other Window 130
Choosing the Other Window 130
Scroll Other Window 130
Scrolling the Other Window 130
Zmacs Commands for Finding Out About Flavors 218
Zmacs Commands for Finding Out About Lisp 217

Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
Finding Out About Zmacs Commands 44
More HELP Commands for Finding Out About Zmacs Commands 46
Overview of Finding Out About Zmacs Commands 44
Finding Out About Zmacs Commands with HELP 44
Finding Out About Zmacs Variables 210
Getting Out of Dired 147
Getting Out of Keystroke Prefixes 42
Getting Out of Minibuffer Prompts 42
Getting Out of Prefixes and Prompts 42
Getting Out of Trouble 42
Overview of Getting Out of Trouble 42
Outputexample 34
Outside FED Command Menu 229
FED Outside World Interface Menu Items 260
Finding Out What an Extended Command Does 45
Finding Out What a Prefix Command Does 45
Example of Finding Out What a Zmacs Command Does 44
Finding Out What a Zmacs Command Does 44
Finding Out What You Have Typed 46
Overview of Changing Case in Zmacs 160
Overview of Commands to Mark Regions 91
Overview of Commenting Lisp Code in Zmacs 171
Overview of Customizing the Zmacs Environment 194
Overview of Deleting Vs. Killing Text 72
Overview of Dired 144
Overview of Evaluating and Compiling Lisp Programs in Zmacs 174
Overview of Finding Out About Zmacs Commands 44
Overview of Getting Out of Trouble 42
Overview of Indentation in Zmacs 162
Overview of Leaving Zmacs 40
Overview of Locating and Replacing Strings Automatically 102
Overview of Moving the Cursor 54
Overview of Numeric Arguments 24
Overview of Searching in Zmacs 98
Overview of Setting the Major Mode 156
Overview of Sorting in Zmacs 112
Overview of the Editor Menu 49
Overview of the Zmacs Manual 4
Overview of Working with Buffers and Files in Zmacs 114
Overview of Zmacs 6
Overview of Zmacs File Manipulation Commands 131
[Map over] Zmail menu item 268
c-X 0 Zmacs command 130

P P P

SELECT P 40
Init File Form: Putting Buffers Into Current Package 213
Setting the Package 138
Set Package (m-X) Zmacs command 138, 143
Packages 138
Backward Page 69

Count Lines Page 48
Forward Page 69
Introduction to Motion by Page 69
Mark Page 92
Motion by Page 69
Next Page 26, 69
Previous Page 26, 69
Commands to Mark Regions by Pages 92
Black pane 239
Drawing pane 227, 262
FED Drawing Pane 227
FED Prompt Pane 228
FED Register Pane 229
FED Sample Pane 227
FED Status Pane 229
Height and width of the drawing pane 256
Mousing on the FED Character Select Pane 263
Mousing on the FED Drawing Pane 262
Mousing on the FED Register Pane 263
Mousing on the FED Sample Pane 262
Prompt pane 228
Register pane 229, 241, 263
Sample pane 227, 235, 253, 262
Setting the Box Size in the FED Drawing Pane 256
Setting the Height and Width of the FED Drawing Pane 256
Size of boxes in the drawing pane 256, 259
Status pane 229
Using the mouse in the character select pane 263
Using the mouse in the drawing pane 262
Using the mouse in the register pane 263
Using the mouse in the sample pane 262
Drawing Pane Menu 228
Backward Paragraph 26, 68
Fill Paragraph 95
Forward Paragraph 26, 68
Introduction to Motion by Paragraph 68
Mark Paragraph 92
Motion by Paragraph 68
Commands to Mark Regions by Paragraphs 92
Example of Commands to Mark Regions by Paragraphs 92
FED Font Parameters Menu 229
Font Parameters menu 229
Mousing on the FED Font Parameters Menu 263
Using the mouse in the Font Parameters menu 263
Find Unbalanced Parentheses 48
Insert Matching parentheses 178
Find Unbalanced Parentheses (m-X) Zmacs command 48
Check Unbalanced Parentheses When Saving Variable 211
Parenthesizing Lisp Expressions in Zmacs 178
Current patch 190
In-progress patch 190
Send mail about patch 190
Add Patch Changed Definitions (m-X) 189
Add Patch Changed Definitions of Buffer (m-X) 188
Active patches 186, 190
Making Patches 186
Inactive patches 190
View Patches (m-X) 190

Add region to patch file 186
Install patch file 190
Patch-file Attribute 143
Patch-File file attribute 143
Set Patch File (m-X) Zmacs command 143
Patching Programs in Zmacs 186
Abort Patch (m-X) 191
Add Patch (m-X) 188
Finish Patch (m-X) 190
Recompile Patch (m-X) 192
Reload Patch (m-X) 192
Resume Patch (m-X) 191
Select Patch (m-X) 190
Start Patch (m-X) 187
Start Private Patch (m-X) 188
Initial patch state 190
In-progress patch state 190
Default Pathnames in Dired 147
P Dired command 153, 268
Entering Peek 40
Performing Operations with Tag Tables 107
Personal default fonts 271
Saving Characters and Pieces of Characters in FED Registers 241
Pixel 196, 227
Zmacs Pl1 Mode 157
Zmacs Electric Pl1 Mode 157
Clear gray plane 239, 260
Getting Things Into the FED Gray Plane 239
Gray plane 228, 239
Merging Characters with the FED Gray Plane 240
Move drawing in the gray plane 239, 260
The FED Gray Plane 239
Gray Plane Menu 228
FED Gray Plane Menu Items 260
Retrieving the Black Plane While Manipulating FED Registers 241
Point 47
Describe Variable At Point 48
Dired move point 149
Editor Window's Cursor and Point 17
Move to Default Previous Point 88
Move to Previous Point 88
Exclamation point (!) line continuation indicator 23, 54
Swap Point And Mark 88
Point and the Region 86
Buffer pointers 86
L2:Move to point mouse click 57
L:Move point mouse click 57
The Point-pdl 87
Clear all points 259
Moving to Previous Points 88
[Clear Points] Font Editor draw mode menu item 235
[Flip Points] Font Editor draw mode menu item 235
[Set Points] Font Editor draw mode menu item 235
Yank Pop 51, 74
Set Pop Mark 88
Default column position 63
Mode Line's *Position-flag* 21
Positioning the Drawing in FED 255

Positioning the Window Around a Definition 56
Displaying the Next Possibility 110
Example of Displaying the Next Possibility 110
Next Possibility 110
Possibility Buffers 110
Set Fill Prefix 95
Prefix character commands 6
Finding Out What a Prefix Command Does 45
Prefix Commands 45
Prefixes 42
Getting Out of Keystroke Prefixes 42
Nullifying prefixes 42
Online documentation for prefixes 45
Getting Out of Prefixes and Prompts 42
Appending, Prepending, and Inserting Text in Zmacs 124
Prepending a Region to a File 124
Prepend To File (m-X) Zmacs command 124
:press-fonts 274
Select Previous Buffer 117
Select Default Previous Buffer 117
Displaying previous keystrokes 46
Moving up to Lisp Comment on Previous Line in Zmacs 172
Deleting the Previous Lisp Expression 79
Previous Page 26, 69
Move to Previous Point 88
Move to Default Previous Point 88
Moving to Previous Points 88
Previous Screen 26
Displaying the Previous Screen 56
Deleting the Previous Sentence 83
Deleting the Previous Word 78
Changing the Default Printer 269
Default printer 271
Default printer for screen copies 271
Printing and Hardcopy Commands 267
Printing messages 268
Start Private Patch (m-X) 188
Procedure for Creating Zmacs Commands with Keyboard Macros 199
Producing Formatted Text 33
Commands for Producing Hardcopy 267
Compiling Lisp Programs in Zmacs 175
Editing Lisp Programs in Zmacs 169
Evaluating and Compiling Lisp Programs in Zmacs 174
Evaluating Lisp Programs in Zmacs 174
Introduction to Editing Lisp Programs in Zmacs 170
Overview of Evaluating and Compiling Lisp Programs in Zmacs 174
Patching Programs in Zmacs 186
Prompt pane 228
FED Prompt Pane 228
Escaping from prompts 42
Getting Out of Minibuffer Prompts 42
Getting Out of Prefixes and Prompts 42
Minibuffer Prompts 42
View File Properties 133
Changing File Properties in Dired 149
Change File Properties (m-X) Zmacs command 133
View File Properties (m-X) Zmacs command 133

Changing the Properties of a File 133
Viewing the Properties of a File 133
Protecting Files From Being Deleted in Dired 152
Protecting Files From Being Reaped in Dired 152
Init File Form: Putting Buffers Into Current Package 213

Q

FUNCTION Q 268, 271
Q Dired command 148
Q Font Editor command 260
Quadruple Numeric Arg 24
Kbd Macro Query 202
Update Attribute List Query 141
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
Query Replace 102
Atom Query Replace 104
Query Replace Last Kill 104
Query Replace Let Binding 104
Query Replace (m-X) Zmacs command 102
Quick Arglist 47
Init File Form: Balanced Quotation Marks and Asterisks 215
c-X Q Zmacs command 202

R

R2:System menu mouse click 57
R Dired command 150
[Read File] Font Editor menu item 258, 260
Reading a File Into a New Buffer 120
Reading a File Into an Existing Buffer 120
Reading and Writing FED Files 257
Reading FED Files 257
Reading font files 258
Down Real Line 26, 63, 81
Up Real Line 26, 63, 81
Init File Form: Setting Goal Column for Real Line Commands 214
Protecting Files From Being Reaped in Dired 152
Reap File (m-X) Zmacs command 136
Dired Complement No Reap Flag 152
Recentering the Window 55
Comparing Recent Versions of Files in Dired 150
Recompile Patch (m-X) 192
Error recovery 42
Redisplaying the Window 55
Introduction to Redisplaying the Window 55
[Reflect] Font Editor menu item 243, 259
Reflecting Drawings in FED 243
Reflecting the drawing 243
Reflecting the drawing 259
REFRESH Font Editor command 261
Region 86
Compile Region 94
Compiling a Region 94

Count Lines Region 48
Creating a Region 87
Deleting a Region 94
Evaluate Region 175
Fill Region 95
Filling a Region 94
Format Region 38
Hardcopying a Region 94
Kill Region 28
Make region 57
Mark region 57
Mark and the Region 86
Point and the Region 86
Save Region 94
Saving a Region 94
Two Windows Showing Region 129
What is a Zmacs Region? 86
Marking a Region From Here to Beginning of Buffer 92
Marking a Region From Here to End of Buffer 92
Creating Two Windows with the Region in Top 129
Indenting Region in Zmacs 165
Region-manipulating Commands 94
Region Marking Mode Variable 211
Compile Region (m-X) Zmacs command 94
Evaluate Region (m-X) Zmacs command 175
Format Region (m-X) Zmacs command 33, 38
Uncomment Region (m-X) Zmacs command 173
Other Region-related Commands 95
Region Right Margin Mode Variable 211
Commands to Mark Regions 91
Exchange Regions 94
Introduction to Regions 86
Overview of Commands to Mark Regions 91
Transposing Regions 94
Commands to Mark Regions by Buffers 92
Commands to Mark Regions by Lisp Expressions 91
Commands to Mark Regions by Pages 92
Commands to Mark Regions by Paragraphs 92
Example of Commands to Mark Regions by Paragraphs 92
Commands to Mark Regions by Words 91
Saving and Inserting Regions in Registers 89
Changing Case of Regions in Zmacs 160
Inserting and Removing Lisp Comments From Regions in Zmacs 173
Working with Regions in Zmacs 85
Appending a Region to a Buffer 124
Appending a Region to a File 124
Prepending a Region to a File 124
Add region to patch file 186
Indenting Region Uniformly in Zmacs 165
Evaluate Region Verbose 175
Creating a Region with Keystrokes 87
Creating a Region with the Mouse 87
Retrieving the Contents of a FED Register 241
Saving a Drawing Into a FED Register 241
Insert text from register into buffer 90
Register pane 229, 241, 263
FED Register Pane 229
Mousing on the FED Register Pane 263

Using the mouse in the register pane 263
Registers 229
Retrieving the Black Plane While Manipulating FED Registers 241
Saving and Inserting Regions in Registers 89
Saving and Moving to Locations in Registers 89
Saving Characters and Pieces of Characters in FED Registers 241
Registers in Zmacs 89
Reindenting Expression in Zmacs 165
Reinitializing Zmacs 12
Reload Patch (m-X) 192
Relocate cursor 57
Inserting and Removing Lisp Comments From Regions in Zmacs 173
Rename Buffer (m-X) Zmacs command 119
[Rename Char] Font Editor menu item 234, 260
Rename File 133
Rename File (m-X) Zmacs command 133
Renaming a File 133
Selecting a FED Character by Renaming Characters 234
Renaming Files 150
Copying and Renaming Files in Dired 150
Renaming the Buffer 119
Reparse Attribute List (m-X) Zmacs command 137
Repeat Last Minibuffer Command 51, 74
Atom Query Replace 104
Query Replace 102
Evaluate and Replace Into Buffer 104
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Query Replace Last Kill 104
Query Replace Let Binding 104
Example of a Tag Tables Replacement Operation 106
Other Types of Replacement Operations in Zmacs 104
Making Global Replacements in Zmacs 102
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
Query Replace (m-X) Zmacs command 102
Replace String (m-X) Zmacs command 102
Searching, Replacing, and Sorting in Zmacs 97
Overview of Locating and Replacing Strings Automatically 102
Locating and Replacing Strings Automatically in Zmacs 102
Reposition Window 56
Re-reading a File Into the Buffer 121
Motion with Respect to the Whole Buffer 70
Cancel response 42
Minibuffer Response Format 51
Minibuffer Response Help 51
More Ways to Enter Minibuffer Responses 51
Moving Rest of Line Down in Zmacs 166
Restoring text 43
Resume Patch (m-X) 191
Setting Generation Retention Count on Files in Dired 152
Retrieving History Elements 74
Retrieving the Black Plane While Manipulating FED Registers 241
Retrieving the Contents of a FED Register 241
Carriage return 6
RETURN completion command 14
Returning to One Window 130

RETURN key in the minibuffer 51
Zmacs Reverse Incremental Search 99
Dired Reverse Undelete 151
Revert Buffer (m-X) Zmacs command 121
Reverting buffers 121, 122
Finding the right command 45
Left and Right Edges of the FED Character Box 237
Region Right Margin Mode Variable 211
R:Menu mouse click 57
[Rotate] Font Editor menu item 243, 259
Rotating Drawings in FED 243
Rotating the drawing 243, 259
Rubout 28
RUBOUT Dired command 151
RUBOUT key 22
RUBOUT Zmacs character 76
RUBOUT Zmacs command 28, 43
c-X RUBOUT Zmacs command 28, 83

S S S

[Set Sample] Font Editor menu item 253, 260
Sample pane 227, 235, 253, 262
FED Sample Pane 227
Mousing on the FED Sample Pane 262
Using the mouse in the sample pane 262
Sample string 227, 253, 260
The FED Sample String 253
What Histories Save 72
M2: Save/Kill/Yank mouse click 57
[Save Char] Font Editor menu item 234, 260
Source Compare Saved Definition 127
Source Compare Merge Saved Definition 127
Save File 30, 121
Save File Zmacs command 31
Save Region 94
Saving a Drawing Into a FED Register 241
Saving a File 31
Saving and Inserting Regions in Registers 89
Saving and Moving to Locations in Registers 89
Saving a Region 94
Saving Buffers 120
Creating and Saving Buffers and Files 30
Description of Creating and Saving Buffers and Files 30
Summary of Creating and Saving Buffers and Files 30
Saving Characters and Pieces of Characters in FED Registers 241
Example 1 of Writing and Saving Keyboard Macros 201
Example 2 of Writing and Saving Keyboard Macros 202
Writing and Saving Keyboard Macros 201
Saving the Buffer Contents to the File 121
Check Unbalanced Parentheses When Saving Variable 211
Scale fraction 239
SCL syntax 140
Scope of the Zmacs Manual 4
Displaying the Next Screen 55
Displaying the Previous Screen 56
Hardcopying the Screen 268

Introduction to the Organization of the Screen 17
Next Screen 26
Organization of the Screen 17
Previous Screen 26
Split Screen 130
Splitting the Screen 130
Default printer for screen copies 271
Scroll bar 255
Scrolling the drawing 255
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Scrolling the Other Window 130
Scroll Other Window 130
SCROLL Zmacs command 26
Zmacs Incremental Search 98
Zmacs Reverse Incremental Search 99
Zmacs String Search 100
Introduction to Tag Tables and Search Domains 106
Tag Tables and Search Domains in Zmacs 106
Searching, Replacing, and Sorting in Zmacs 97
Searching for Appropriate Commands 45
Searching for Appropriate Zmacs Commands 45
Method for Searching for Appropriate Zmacs Commands 45
Searching in Zmacs 98
Overview of Searching in Zmacs 98
Example of a Search String for HELP A 46
Using Two Windows, Select Bottom 129
Select Buffer 30, 117
SELECT C 40
SELECT D 40
Select Default Previous Buffer 117
SELECT E 12, 40
Entering Zmacs with SELECT E 12
Selected buffer 116
SELECT F 40
SELECT I 40
Selecting, Listing, and Examining Zmacs Buffers 116
Selecting a Character in FED 233
Selecting a FED Character by Creating a New Character 233
Selecting a FED Character by Renaming Characters 234
Selecting a FED Character From the Character Select Menu 233
Selecting a FED Character From the [Show Font] Display 233
Selecting a FED Character with the C Command 233
Selecting a font 260
Selecting a Font in FED 230
SELECT key 40
Leaving Zmacs with the SELECT Key 40
SELECT L 40
SELECT M 40
Character Select menu 229
FED Character Select Menu 229
Selecting a FED Character From the Character Select Menu 233
SELECT N 40

SELECT P 40
Mousing on the FED Character Select Pane 263
Using the mouse in the character select pane 263
Select Patch (m-X) 190
Select Previous Buffer 117
SELECT T 40
Using Two Windows, Select Top 129
SELECT X 40
Semicolon (;) comment indicator 171
Send mail about patch 190
zwei: ***send-mail-about-patch*** 190
Mouse Sensitivities in FED 262
Backward Sentence 26, 62
Backward Kill Sentence 28
Deleting the Current Sentence 83
Deleting the Previous Sentence 83
Forward Sentence 62
Kill Sentence 28, 83
Motion by Sentence 62
Description of Zmacs Sentence Delimiters 62, 83
Deleting Sentences 83
Introduction to Deleting Sentences 83
zwei: ***set-attribute-update-list*** global variable 141, 143
Set Backspace (m-X) Zmacs command 141, 143
Set Base (m-X) Zmacs command 141, 143
Set commands 143
Set commands for file and buffer attributes 143
Other Set Commands for File and Buffer Attributes 140
:set-cursorpos method of **si:make-hardcopy-stream** 274
si: **set-default-hardcopy-device** function 271
Set Fill Column 196
Set Fill Prefix 95
:set-font method of **si:make-hardcopy-stream** 274
Set Fonts (m-X) Zmacs command 141, 143
Set Goal Column 63
Set Lowercase (m-X) Zmacs command 142, 143
Set Nofill (m-X) Zmacs command 142, 143
Set Package (m-X) Zmacs command 138, 143
Set Patch File (m-X) Zmacs command 143
[Set Points] Font Editor draw mode menu item 235
Set Pop Mark 88
[Set Sample] Font Editor menu item 253, 260
si: **set-screen-hardcopy-device** function 271
Set Syntax (m-X) 140
Settable Zmacs Variables 211
Set Tab Width (m-X) Zmacs command 143
Setting/Popping the Mark 88
Setting Buffer Attributes 141
Init File Form: Setting Default Major Mode 213
Setting Editor Variables in Init Files 213
Init File Form: Setting Find File Not to Create New Files 214
Setting Generation Retention Count on Files in Dired 152
Init File Form: Setting Goal Column for Real Line Commands 214
Setting Key Bindings in Init Files 215
Setting Mode Hooks in Init Files 214
How to Specify Zmacs Variable Settings 210

Base and Syntax Default Settings for Lisp 31, 120, 156, 170
Setting the Box Size in the FED Drawing Pane 256
Setting the Height and Width of the FED Drawing Pane 256
Setting the Key 208
Setting the Lisp Comment Column in Zmacs 172
Overview of Setting the Major Mode 156
Setting the Package 138
Setting the Syntax for Symbolics Common Lisp 140
Setting the Zmacs Major Mode 155
Setting Variables 211
Set Variable 212
Set Variable (m-X) Zmacs command 212
Set Visited File Name (m-X) Zmacs command 122
Set Vsp (m-X) Zmacs command 143
Backward Sexp 65
Backward Kill Sexp 28, 79
Forward Sexp 65
Kill Sexp 28, 79
Mark Sexp 91
Exchange Sexps 79
S Font Editor command 260
Shift keys 6
Init File Form: Electric Shift Lock in Lisp Mode 214
Show Directory/View Directory 132
Show Directory (m-X) Zmacs command 132
Show Documentation 47
Show File (m-X) Zmacs command 132
Selecting a FED Character From the [Show Font] Display 233
Mousing on the FED List Fonts and Show Font Displays 263
[Show Font] Font Editor menu item 227, 233, 234, 260
Using the mouse with [Show Font] Font Editor menu item 234, 263
Two Windows Showing Region 129
Showing the Mark 88
:show-line method of **si:make-hardcopy-stream** 275
:show-rectangle method of **si:make-hardcopy-stream** 274
:default-font keyword to **si:*hardcopy-default-fonts*** 271
:header-font keyword to **si:*hardcopy-default-fonts*** 271
si:*hardcopy-default-fonts* variable 271
si:hardcopy-from-stream function 274
si:hardcopy-text-file function 273
:convert-to-device-units method of **si:make-hardcopy-stream** 275
:eject-page method of **si:make-hardcopy-stream** 274
:set-cursorpos method of **si:make-hardcopy-stream** 274
:set-font method of **si:make-hardcopy-stream** 274
:show-line method of **si:make-hardcopy-stream** 275
:show-rectangle method of **si:make-hardcopy-stream** 274
:un-relative-coordinates method of **si:make-hardcopy-stream** 275
si:make-hardcopy-stream function 274
si:set-default-hardcopy-device function 271
si:set-screen-hardcopy-device function 271
List the last sixty commands 46
List the last sixty keystrokes 46
Changing Window Size 129
[Grid Size] Font Editor menu item 256, 259

Setting the Box Size in the FED Drawing Pane 256
Size of boxes in the drawing pane 256, 259
Using Keyboard Macros to Sort 203
Zmacs Sorting Commands 112
Sorting in Zmacs 112
Overview of Sorting in Zmacs 112
Searching, Replacing, and Sorting in Zmacs 97
Sort Via Keyboard Macros 203
Finding source code 170
Introduction to Locating Source Code in Zmacs 180
Editing the source code of a function 12
Locating Source Code to Edit in Zmacs 180
Source Compare 125, 131
Example of a Source Compare 125
Source Compare Installed Definition 127
Source Compare Merge 125
Source Compare Merge Installed Definition 128
Source Compare Merge (m-X) Zmacs command 125
Source Compare Merge Newest Definition 127
Source Compare Merge Saved Definition 127
Source Compare (m-X) Zmacs command 125
Source Compare Newest Definition 127
Source Compare Saved Definition 127
SPACE 14
Delete Horizontal Space 28
SPACE completion command 14
SPACE Dired command 149
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: White Space in Lisp Code 215
HELP SPACE Zmacs command 14
Vertical spacing 143
Moving to a Specified Line 56
Specifying and Listing Tag Tables 106
Creating Two Windows, Specifying Other Contents 129
How to Specify Zmacs Variable Settings 210
Draw a cubic spline 259
[Draw Spline] Font Editor menu item 246, 259
Split Screen 130
Splitting the Screen 130
Init File Form: c-m-L on the SQUARE Key 215
Dired Srccom 150
Standard comtab 208
Standard TV Fonts 223
Getting Started in Zmacs 11
Starting a Keyboard Macro 200
Starting a New Line 22
Starting Zmacs 12
Start Kbd Macro 199
Erase backward to start of line 82
Start Patch (m-X) 187
Start Private Patch (m-X) 188
Initial patch state 190
In-progress patch state 190
Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
Status line documentation 255
Checking the Status of Hardcopy Devices 269
Status pane 229

FED Status Pane 229
[Stretch] Font Editor menu item 259
Stretching a character 259
Stretching a Drawing Horizontally in FED 250
Stretching a Drawing Vertically in FED 250
Stretching and Contracting Drawings in FED 246
Example of Listing Variables by Matching a String 211
Listing Variables by Matching a String 210
Sample string 227, 253, 260
The FED Sample String 253
Example of a Search String for HELP A 46
Replace String (M-X) Zmacs command 102
Overview of Locating and Replacing Strings Automatically 102
Locating and Replacing Strings Automatically in Zmacs 102
Zmacs String Search 100
Text subheading 37
FED, the Subsystem 227
Dired Command Summary 146
Zmacs Help Command Summary 217
Summary of Creating and Saving Buffers and Files 30
Summary of Cursor Movement 54
Summary of Erasing Text 28
Summary of Moving the Cursor 26
Summary of Zmacs Minor Modes 196
Support Buffers 109
Supported file formats 258
Getting Things Into Gray with [Swap Gray] 239
[Swap Gray] Font Editor menu item 239, 260
Swap Point And Mark 88
Setting the Syntax for Symbolics Common Lisp 140
Default syntax 140
SCL syntax 140
Zetalisp syntax 140
Base and Syntax Defaults 139
Base and Syntax Default Settings for Lisp 31, 120, 156, 170
Setting the Syntax for Symbolics Common Lisp 140
Set Syntax (M-X) 140
Hardcopy System 265
Entering File System Editor 40
Hardcopying From the File System Editor 268
Hardcopying From the System Menu 267
Leaving Zmacs Via the System Menu 40
R2: System menu mouse click 57
C-X S Zmacs command 30

T T T

@t 34
SELECT T 40
TAB 163
@tabclear 37
@tabdivide 37
TAB in **loop** macro 162
Command table 208
Char-exists table 223
Char-exists Table Font Attribute 223
Command tables 7

Introduction to Zmacs Command Tables 7
Performing Operations with Tag Tables 107
Specifying and Listing Tag Tables 106
Introduction to Tag Tables and Search Domains 106
Tag Tables and Search Domains in Zmacs 106
Example of a Tag Tables Replacement Operation 106
Example 1 of Making Tables Using Keyboard Macros 206
Example 2 of Making Tables Using Keyboard Macros 206
Making Tables Using Keyboard Macros 205
How Tag Tables Work 106
@tabset 37
Tab-width Attribute 143
Tab-Width file attribute 143
Set Tab Width (m-X) Zmacs command 143
Performing Operations with Tag Tables 107
Specifying and Listing Tag Tables 106
Introduction to Tag Tables and Search Domains 106
Tag Tables and Search Domains in Zmacs 106
Example of a Tag Tables Replacement Operation 106
How Tag Tables Work 106
Zmacs Teco Mode 157
zwei: ***temp-file-type-list*** variable 135
Entering Terminal 40
Boldface text 34
Center text 34
Deleting Vs. Killing Text 72
Description text 34
Description of Erasing Text 28
Display text 34
Enumerate text 34
Erasing text 76
Example text 34
Format text 34
Inserting Text 22
Introduction to Erasing Text 28
Introduction to Inserting Text 22
Itemize text 34
Marking text 91
Overview of Deleting Vs. Killing Text 72
Producing Formatted Text 33
Restoring text 43
Summary of Erasing Text 28
Transposing Lines of Text 82
Verbatim text 34
Zmacs Commands for Formatting Text 33
Getting Text Back 43
Text example 37
Text formatting 37
Basic Text Formatting Commands 37
Zmacs text formatting commands 37
Basic Text Formatting Environments 34
Introduction to Text Formatting in Zmacs 33
Insert text from register into buffer 90
Text heading 37
Appending, Prepending, and Inserting Text in Zmacs 124
Deleting and Transposing Text in Zmacs 71
Init File Form: Auto Fill in Text Mode 214
Zmacs Text Mode 156

Text subheading 37
Example of Filling Text with Auto Fill Minor Mode 195
Finding Files That Have Not Been Backed up in Dired 152
How They Work 137
M:Mark thing mouse click 57
Getting Things Into Gray with [Gray Char] 239
Getting Things Into Gray with [Swap Gray] 239
Getting Things Into the FED Gray Plane 239
New Line with This Indentation in Zmacs 166
Append To Buffer 124
Append To File (m-X) Zmacs command 124
Prepend To File (m-X) Zmacs command 124
Toggle 195
Creating Two Windows with the Region in Top 129
Using Two Windows, Select Top 129
Top Edge of the FED Character Box 237
Abort At Top Level 42
Motion Among Top-level Expressions 66
Trace 48
Trace (m-X) Zmacs command 48
FED Configuration and Drawing Transformation 259
Transformations on Characters in FED 243
Transposing Characters 76
Deleting and Transposing Characters 76
Deleting and Transposing Lines 81
Introduction to Deleting and Transposing Lines 81
Transposing Lines of Text 82
Transposing Lisp Expressions 79
Deleting and Transposing Lisp Expressions 79
Introduction to Deleting and Transposing Lisp Expressions 79
Transposing Regions 94
Deleting and Transposing Text in Zmacs 71
Transposing Words 78
Deleting and Transposing Words 78
Introduction to Deleting and Transposing Words 78
Getting Out of Trouble 42
Overview of Getting Out of Trouble 42
Attributes of TV Fonts 221
Standard TV Fonts 223
:tv-fonts 274
Two window mode 129
Two Windows 129
Modified Two Windows 129
Using Two Windows 129
View Two Windows 129
Using Two Windows, Select Bottom 129
Using Two Windows, Select Top 129
Creating Two Windows, Specifying Other Contents 129
Two Windows Showing Region 129
Creating Two Windows with the Region in Top 129
Finding Out What You Have Typed 46
Typeout 17
Editor Window's Typeout 17
Typeout window 17, 44
Canonical types 134
File Types and Zmacs Major Modes 198
Other Types of Replacement Operations in Zmacs 104
Typical use of Zmacs 170

Correcting Typos 22
c-X T Zmacs command 94

U U U

Help U 46
U Dired command 151
Find Unbalanced Parentheses 48
Find Unbalanced Parentheses (m-X) Zmacs command 48
Check Unbalanced Parentheses When Saving Variable 211
Uncomment Region (m-X) Zmacs command 173
Dired Undelete 151
Dired Reverse Undelete 151
Undo all changes to buffer 121
Dired Next Undumped 153
Indenting Region Uniformly in Zmacs 165
Unknown attribute 138
:un-relative-coordinates method of **si:make-hardcopy-stream** 275
Motion up and Down Nesting Levels 66
Update Attribute List (m-X) Zmacs command 138
Update Attribute List Query 141
Updating the Dired Display 145
Finding Files That Have Not Been Backed up in Dired 152
Up Line 63, 81
Backward up List 66
Forward up List 66
Kill Backward Up List (c-m-X) Zmacs command 79
Up Real Line 26, 63, 81
Moving up to Lisp Comment on Previous Line in Zmacs 172
User-defined Zmacs Major Modes 198
HELP U Zmacs command 14, 46

V V V

Help V 46
Check Unbalanced Parentheses When Saving Variable 211
Definition of a Zmacs Variable 210
Describe Variable 210
fs:file-type-mode-alist Lisp variable 198
ibase global variable 141, 143
One Window Default Variable 211
Region Marking Mode Variable 211
Region Right Margin Mode Variable 211
Set Variable 212
si:*hardcopy-default-fonts* variable 271
zwei:*file-versions-kept* variable 135
zwei:*major-mode-translations* Lisp variable 198
zwei:*set-attribute-update-list* global variable 141, 143
zwei:*temp-file-type-list* variable 135
Variable Apropos 210
Variable Apropos Zmacs command 210
Describe Variable At Point 48
Describe Variable (m-X) Zmacs command 210
Set Variable (m-X) Zmacs command 212
Describing Zmacs Variables 210
Finding Out About Zmacs Variables 210

List Variables 210
Listing Zmacs Variables 210
Settable Zmacs Variables 211
Setting Variables 211
Zmacs variables 46
Example of Listing Variables by Matching a String 211
Listing Variables by Matching a String 210
How to Specify Zmacs Variable Settings 210
Setting Editor Variables in Init Files 213
List Variables (m-X) Zmacs command 210
Variable-width fonts 222
V Dired command 150
Verbatim text 34
Evaluate Region Verbose 175
Mode Line's *Version* 21
Minor version number 186
Comparing/Merging Current/Installed Versions 127
Comparing/Merging Current/Newest Versions 127
Comparing/Merging Current/Saved Versions 127
Comparing file versions 150
Deleting Multiple Versions 135
File versions 114
Deleting Multiple File Versions in Dired 151
Comparing Recent Versions of Files in Dired 150
Contracting a Drawing Vertically in FED 250
Moving the Drawing Horizontally and/or Vertically in FED 255
Scrolling the Drawing Horizontally and/or Vertically in FED 255
Stretching a Drawing Vertically in FED 250
Vertical spacing 143
V Font Editor command 253, 260
Sort Via Keyboard Macros 203
Leaving Zmacs Via the System Menu 40
View Buffer 119
View Buffer (m-X) Zmacs command 119
View Directory (m-X) Zmacs command 132
Dired View File 150
View File (m-X) Zmacs command 132
View File Properties 133
View File Properties (m-X) Zmacs command 133
[Center View] Font Editor menu item 255, 259
[Move View] Font Editor menu item 255, 259
Viewing a Buffer 119
Viewing a File 132
Viewing a Keyboard Macro 200
Viewing and Altering a Character in the FED Character Box 237
Viewing and Editing File Contents in Dired 149
Viewing File Attributes in Dired 149
Viewing the Editor Command History 73
Viewing the Kill History 73
Viewing the Properties of a File 133
View Kbd Macro (m-X) Zmacs command 200
View Patches (m-X) 190
View Two Windows 129
Set Visited File Name (m-X) Zmacs command 122
Visit File 120
Deleting Vs. Killing Text 72
Overview of Deleting Vs. Killing Text 72

Vsp Attribute 143
Vsp file attribute 143
Set Vsp (m-X) Zmacs command 143
c-HELP V Zmacs command 210
c-m-? V Zmacs command 210
c-X V Zmacs command 119
HELP V Zmacs command 14, 46, 210

W W W

Help W 46
Lisp Compiler Warnings 176
Warnings about file attribute lists 138
More Ways to Enter Minibuffer Responses 51
Finding Out What an Extended Command Does 45
Finding Out What a Prefix Command Does 45
Example of Finding Out What a Zmacs Command Does 44
Finding Out What a Zmacs Command Does 44
What Histories Save 72
What is a Zmacs Region? 86
What the Lines Mean in the FED Character Box 237
Finding Out What You Have Typed 46
Check Unbalanced Parentheses When Saving Variable 211
Where Am I 47
Fast Where Am I 47
Using the CONTROL key while drawing characters 235
Using the META key while drawing characters 235
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
Retrieving the Black Plane While Manipulating FED Registers 241
Init File Form: Fixing White Space for Kill/Yank Commands 214
Init File Form: White Space in Lisp Code 215
Motion with Respect to the Whole Buffer 70
Wide Configuration 227
Blinker width 223
Character width 222, 238
Blinker *Width* and *Blinker Height* Font Attributes 223
Character *Width* Font Attribute 222
Set Tab Width (m-X) Zmacs command 143
Height and width of the drawing pane 256
Setting the Height and Width of the FED Drawing Pane 256
Choosing the Other Window 130
Grow Window 129
Introduction to Redisplaying the Window 55
One Window 130
Other Window 130
Recentering the Window 55
Redisplaying the Window 55
Reposition Window 56
Returning to One Window 130
Scrolling the Other Window 130
Scroll Other Window 130
Typeout window 17, 44
Wraparound Lines in the Editor Window 54
Zmacs Editor Window 17
Editor Window's Buffer 17

Editor Window's Cursor and Point 17
Editor Window's Typeout 17
The Editor Window and the Buffer 54
Positioning the Window Around a Definition 56
Zmacs Window Commands 129
One Window Default Variable 211
Two window mode 129
Modified Two Windows 129
Multiple windows 114
Two Windows 129
Using Two Windows 129
View Two Windows 129
Using Two Windows, Select Bottom 129
Using Two Windows, Select Top 129
Creating Two Windows, Specifying Other Contents 129
Changing Window Size 129
Two Windows Showing Region 129
Creating Two Windows with the Region in Top 129
Backward Word 26, 61
Backward Kill Word 28, 78
Deleting the Current Word 78
Deleting the Previous Word 78
Forward Word 26, 61
Kill Word 28, 78
Motion by Word 61
Commands to Mark Regions by Words 91
Deleting and Transposing Words 78
Exchange Words 78
Introduction to Deleting and Transposing Words 78
Transposing Words 78
Changing Case of Words in Zmacs 160
How Tag Tables Work 106
How They Work 137
How Zmacs Keyboard Macros Work 199
How Zmacs Minor Modes Work 195
Working with Buffers and Files in Zmacs 114
Overview of Working with Buffers and Files in Zmacs 114
Working with Regions in Zmacs 85
How Key Bindings Work: the Comtab 208
FED Outside World Interface Menu Items 260
Wraparound Lines 54
Wraparound Lines in the Editor Window 54
Wrapping Lines 23
Write File 30, 121
[Write File] Font Editor menu item 258, 260
Write File Zmacs command 32
Writing and Saving Keyboard Macros 201
Example 1 of Writing and Saving Keyboard Macros 201
Example 2 of Writing and Saving Keyboard Macros 202
Writing FED Files 258
Reading and Writing FED Files 257
Writing font files 258
Writing the Buffer Contents to a File 121
c-X W Zmacs command 30
HELP W Zmacs command 14
HELP W Zmacs command 46

X X X

SELECT X 40

Y Y Y

Yank 74
c-0 c-m-Y yank command 15
c-m-Y yank command 15
c-0 c-Y yank command 15
c-Y yank command 15
m-Y yank command 15, 51, 74
Yanking 15, 43
Introduction to Yanking 15
Yanking in the command history 15
Yanking in the kill history 15
Yanking in the Minibuffer 51
Yank Pop 51, 74
Finding Out What You Have Typed 46

Z Z Z

Zetalisp syntax 140
+ flag in Zmacs 115
Aligning Indentation in Zmacs 165
Appending, Prepending, and Inserting Text in Zmacs 124
Buffer and File Attributes in Zmacs 137
Centering the Current Line in Zmacs 164
Changing Case and Indentation in Zmacs 159
Changing Case in Zmacs 160
Changing Case of Buffers in Zmacs 160
Changing Case of Regions in Zmacs 160
Changing Case of Words in Zmacs 160
Commenting Lisp Code in Zmacs 171
Comparing Files and Buffers in Zmacs 125
Compiling Lisp Programs in Zmacs 175
Creating a New Indented Lisp Comment Line in Zmacs 172
Deleting and Transposing Text in Zmacs 71
Deleting Blank Line in Zmacs 167
Deleting Indentation in Zmacs 166
Dired Mode in Zmacs 144
Editing Lisp Programs in Zmacs 169
Entering Zmacs 12, 40
Evaluating and Compiling Lisp Programs in Zmacs 174
Evaluating Lisp Programs in Zmacs 174
Expanding Lisp Expressions in Zmacs 179
Getting Help in Zmacs 41
Getting Started in Zmacs 11
Going Back to First Indented Character in Zmacs 165
Hardcopying From Zmacs 267
Indentation in Zmacs 162
Indenting Current Line in Zmacs 162
Indenting for Lisp Comment in Zmacs 171
Indenting New Line in Zmacs 164
Indenting Region in Zmacs 165
Indenting Region Uniformly in Zmacs 165
Inserting and Removing Lisp Comments From Regions in

Zmacs 173
Inserting Blank Line in Zmacs 167
Introduction to Zmacs 6
Introduction to Customizing Zmacs 194
Introduction to Editing Lisp Programs in Zmacs 170
Introduction to Entering Zmacs 12
Introduction to Locating Source Code in Zmacs 180
Introduction to Text Formatting in Zmacs 33
Invoking Zmacs 12
Killing a Lisp Comment in Zmacs 171
Leaving Zmacs 40
Locating and Replacing Strings Automatically in Zmacs 102
Locating Source Code to Edit in Zmacs 180
Making Global Replacements in Zmacs 102
Manipulating Buffers and Files in Zmacs 113
Mouse Documentation Line in Zmacs 57
Moving Down to Lisp Comment on Next Line in Zmacs 172
Moving Rest of Line Down in Zmacs 166
Moving the Cursor in Zmacs 53
Moving up to Lisp Comment on Previous Line in Zmacs 172
New Line with This Indentation in Zmacs 166
Other Types of Replacement Operations in Zmacs 104
Overview of Zmacs 6
Overview of Changing Case in Zmacs 160
Overview of Commenting Lisp Code in Zmacs 171
Overview of Evaluating and Compiling Lisp Programs in Zmacs 174
Overview of Indentation in Zmacs 162
Overview of Leaving Zmacs 40
Overview of Searching in Zmacs 98
Overview of Sorting in Zmacs 112
Overview of Working with Buffers and Files in Zmacs 114
Parenthesizing Lisp Expressions in Zmacs 178
Patching Programs in Zmacs 186
Querying While Making Global Replacements in Zmacs 102
Querying While Making Multiple Global Replacements in Zmacs 103
Registers in Zmacs 89
Reindenting Expression in Zmacs 165
Reinitializing Zmacs 12
Searching, Replacing, and Sorting in Zmacs 97
Searching in Zmacs 98
Setting the Lisp Comment Column in Zmacs 172
Sorting in Zmacs 112
Starting Zmacs 12
Tag Tables and Search Domains in Zmacs 106
Typical use of Zmacs 170
Using the mouse to enter Zmacs 12
Working with Buffers and Files in Zmacs 114
Working with Regions in Zmacs 85
Zmacs Commands for Finding Out About the State of Zmacs 217
Zmacs Bolio Mode 157
Current Zmacs Buffer 116
Zmacs Buffer and File Names 114
Zmacs Buffer Commands 117
Zmacs Buffer History 116
Selecting, Listing, and Examining Zmacs Buffers 116

RUBOUT Zmacs character 76
ABORT Zmacs command 42
Append To File (m-X) Zmacs command 124
Arglist (m-X) Zmacs command 48
c-% Zmacs command 102
c-; Zmacs command 171
c-= Zmacs command 47
c-A Zmacs command 26, 63, 81
c-B Zmacs command 26, 61
c-D Zmacs command 28, 43, 76
c-E Zmacs command 26, 63, 81
c-F Zmacs command 26, 61
c-G Zmacs command 42
Change File Properties (m-X) Zmacs command 133
c-HELP V Zmacs command 210
c-K Zmacs command 28, 81
c-L Zmacs command 55
Clean Directory (m-X) Zmacs command 136
CLEAR-INPUT Zmacs command 82
c-m-(Zmacs command 66
c-m-) Zmacs command 66
c-m-@ Zmacs command 91
c-m-] Zmacs command 67
c-m-[Zmacs command 66
c-m-; Zmacs command 171
c-m-? V Zmacs command 210
c-m-A Zmacs command 66
c-m-B Zmacs command 65
c-m-D Zmacs command 66
c-m-E Zmacs command 67
c-m-F Zmacs command 65
c-m-H Zmacs command 91
c-m-K Zmacs command 28, 79
c-m-L Zmacs command 117
c-m-N Zmacs command 65
c-m-O Zmacs command 166
c-m-P Zmacs command 65
c-m-Q Zmacs command 165
c-m-R Zmacs command 56
c-m-RUBOUT Zmacs command 28, 79
c-m-sh-E Zmacs command 175
c-m-SPACE Zmacs command 88
c-m-T Zmacs command 79
c-m-U Zmacs command 66
c-m-V Zmacs command 130
c-m-Z Zmacs command 175
c-m-\ Zmacs command 165
c-m-^ Zmacs command 166
c-N Zmacs command 26, 63, 81, 199
c-O Zmacs command 167
Compile Region (m-X) Zmacs command 94
Copy File (m-X) Zmacs command 134
c-P Zmacs command 26, 63, 81
Create Directory (m-X) Zmacs command 131
Create Link (m-X) Zmacs command 135
c-sh-A Zmacs command 47
c-sh-C Zmacs command 94
c-sh-D Zmacs command 47

c-sh-E Zmacs command 175
c-sh-V Zmacs command 48
c-SPACE Zmacs command 88
c-T Zmacs command 76
c-U Zmacs command 24
c-V Zmacs command 26, 55
c-W Zmacs command 28, 94
c-X Zmacs command 199
c-X) Zmacs command 199
c-X] Zmacs command 26, 69
c-X [Zmacs command 26, 69
c-X 1 Zmacs command 130
c-X 2 Zmacs command 129
c-X 3 Zmacs command 129
c-X 4 Zmacs command 129
c-X 8 Zmacs command 129
c-X ; Zmacs command 172
c-X = Zmacs command 47
c-X A Zmacs command 124
c-X B Zmacs command 30, 31, 117
c-X c-; Zmacs command 173
c-X c-B Zmacs command 115, 118
c-X c-D Zmacs command 132
c-X c-F Zmacs command 31, 32
c-X c-I Zmacs command 165
c-X c-L Zmacs command 160
c-X c-m-L Zmacs command 117
c-X c-m-SPACE Zmacs command 88
c-X c-N Zmacs command 63
c-X c-O Zmacs command 28, 167
c-X c-P Zmacs command 92
c-X c-S Zmacs command 31, 121
c-X c-T Zmacs command 82
c-X c-U Zmacs command 160
c-X c-V Zmacs command 120
c-X c-W Zmacs command 32, 121
c-X c-X Zmacs command 88
c-X D Zmacs command 144
c-X E Zmacs command 200
c-X F Zmacs command 30, 196
c-X L Zmacs command 48
c-X O Zmacs command 130
c-X Q Zmacs command 202
c-X RUBOUT Zmacs command 28, 83
c-X S Zmacs command 30
c-X T Zmacs command 94
c-X V Zmacs command 119
c-X W Zmacs command 30
c-X ^ Zmacs command 129
Deinstall Macro (m-X) Zmacs command 204
Delete File (m-X) Zmacs command 135
Describe Variable (m-X) Zmacs command 210
Dired (m-X) Zmacs command 144
Edit Buffers (m-X) Zmacs command 118
Edit Directory (m-X) Zmacs command 144
End Kbd Macro Zmacs command 199
Evaluate and Replace Into Buffer (m-X) Zmacs command 175
Evaluate Buffer (m-X) Zmacs command 175

Evaluate Changed Definitions (m-X) Zmacs command 175
Evaluate Changed Definitions of Buffer (m-X) Zmacs command 175
Evaluate Into Buffer (m-X) Zmacs command 175
Evaluate Region (m-X) Zmacs command 175
Find File Zmacs command 31, 32
Find File In Fundamental Mode (m-X) Zmacs command 122
Find Unbalanced Parentheses (m-X) Zmacs command 48
Format Buffer (m-X) Zmacs command 33, 38
Format File (m-X) Zmacs command 33, 38
Format Region (m-X) Zmacs command 33, 38
Hardcopy Buffer (m-X) Zmacs command 119, 267
HELP ? Zmacs command 14
HELP A Zmacs command 14, 45
HELP C Zmacs command 14, 45
HELP D Zmacs command 14, 45
HELP L Zmacs command 14, 46
HELP SPACE Zmacs command 14
HELP U Zmacs command 14, 46
HELP V Zmacs command 14, 46, 210
HELP W Zmacs command 14
HELP W Zmacs command 46
Insert Buffer (m-X) Zmacs command 124
Insert File (m-X) Zmacs command 124
Install Command (m-X) Zmacs command 209
Install Macro (m-X) Zmacs command 203
Kill Backward Up List (c-m-X) Zmacs command 79
LINE Zmacs command 164
Lisp Mode (m-X) Zmacs command 170
List Buffers Zmacs command 115
List Changed Definitions of Buffer (m-X) Zmacs command 183
List Definitions (m-X) Zmacs command 183
List Files (m-X) Zmacs command 131
List Fonts (m-X) Zmacs command 223, 271
List Variables (m-X) Zmacs command 210
m-@ Zmacs command 91
m-] Zmacs command 26, 68
m-[Zmacs command 26, 68
m-) Zmacs command 67
m-% Zmacs command 102
m-; Zmacs command 171
m-< Zmacs command 26, 70
m-= Zmacs command 48
m-> Zmacs command 26, 70
m-A Zmacs command 26
m-B Zmacs command 26, 61
m-C Zmacs command 160
m-D Zmacs command 28, 78
m-E Zmacs command 62
m-ESCAPE Zmacs command 175
m-F Zmacs command 26, 61
m-H Zmacs command 92
m-K Zmacs command 28, 83
m-L Zmacs command 160
m-LINE Zmacs command 172
m-N Zmacs command 172
m-O Zmacs command 166
m-P Zmacs command 172
m-R Zmacs command 56

m-RUBOUT Zmacs command 28, 78
m-S Zmacs command 164
m-SCROLL Zmacs command 26, 56
m-sh-D Zmacs command 47
m-sh-E Zmacs command 175
m-T Zmacs command 78
m-U Zmacs command 160
m-V Zmacs command 26, 56
m-W Zmacs command 94
m-Z Zmacs command 176
m-\ Zmacs command 28
m-^ Zmacs command 28, 166
Name Last Kbd Macro (m-X) Zmacs command 203
Prepend To File (m-X) Zmacs command 124
Query Replace (m-X) Zmacs command 102
Reap File (m-X) Zmacs command 136
Rename Buffer (m-X) Zmacs command 119
Rename File (m-X) Zmacs command 133
Reparse Attribute List (m-X) Zmacs command 137
Replace String (m-X) Zmacs command 102
Revert Buffer (m-X) Zmacs command 121
RUBOUT Zmacs command 28, 43
Save File Zmacs command 31
SCROLL Zmacs command 26
Set Backspace (m-X) Zmacs command 141, 143
Set Base (m-X) Zmacs command 141, 143
Set Fonts (m-X) Zmacs command 141, 143
Set Lowercase (m-X) Zmacs command 142, 143
Set Nofill (m-X) Zmacs command 142, 143
Set Package (m-X) Zmacs command 138, 143
Set Patch File (m-X) Zmacs command 143
Set Tab Width (m-X) Zmacs command 143
Set Variable (m-X) Zmacs command 212
Set Visited File Name (m-X) Zmacs command 122
Set Vsp (m-X) Zmacs command 143
Show Directory (m-X) Zmacs command 132
Show File (m-X) Zmacs command 132
Source Compare Merge (m-X) Zmacs command 125
Source Compare (m-X) Zmacs command 125
Trace (m-X) Zmacs command 48
Uncomment Region (m-X) Zmacs command 173
Update Attribute List (m-X) Zmacs command 138
Variable Apropos Zmacs command 210
View Buffer (m-X) Zmacs command 119
View Directory (m-X) Zmacs command 132
View File (m-X) Zmacs command 132
View File Properties (m-X) Zmacs command 133
View Kbd Macro (m-X) Zmacs command 200
Write File Zmacs command 32
Example of Finding Out What a Zmacs Command Does 44
Finding Out What a Zmacs Command Does 44
Finding Out About Zmacs Commands 44
General Information-giving Zmacs Commands 46
Introduction to Zmacs Commands 6
Method for Searching for Appropriate Zmacs Commands 45
More HELP Commands for Finding Out About Zmacs Commands 46
Mouse-sensitive Zmacs commands 57
Overview of Finding Out About Zmacs Commands 44

Searching for Appropriate Zmacs Commands 45
Zmacs Commands for Finding Out About Flavors 218
Zmacs Commands for Finding Out About Lisp 217
Zmacs Commands for Finding Out About the State of Buffers 217
Zmacs Commands for Finding Out About the State of Zmacs 217
Zmacs Commands for Formatting Text 33
Zmacs Commands for Interacting with Lisp 218
Finding Out About Zmacs Commands with HELP 44
Creating New Zmacs Commands with Keyboard Macros 199
Procedure for Creating Zmacs Commands with Keyboard Macros 199
Introduction to Zmacs Command Tables 7
zwei: ***zmacs-comtab*** 201
Zmacs Echo Area 19
The Zmacs Edit Callers Commands 183
The Zmacs Edit Definition Commands 180
Zmacs Editor Window 17
Zmacs Electric Pl1 Mode 157
Customizing the Zmacs Environment 193
Overview of Customizing the Zmacs Environment 194
Introduction to Zmacs Extended Commands 7
Zmacs File Manipulation Commands 131
Overview of Zmacs File Manipulation Commands 131
Zmacs Format Commands 38
Zmacs Fortran Mode 156
Zmacs Fundamental Mode 156
Zmacs Help 14
Introduction to Zmacs Help 14
Zmacs Help Command Summary 217
Zmacs Incremental Search 98
Customizing Zmacs in Init Files 213
Introduction to Customizing Zmacs in Init Files 213
Zmacs Key Bindings 208
Definition of a Zmacs Keyboard Macro 199
How Zmacs Keyboard Macros Work 199
Introduction to Zmacs Keystrokes 6
Zmacs Lisp Mode 156
The Zmacs List Definition Commands 182
Zmacs Macsyma Mode 157
Zmacs Major Editing Modes 156
Setting the Zmacs Major Mode 155
Zmacs Major Modes 198
File Types and Zmacs Major Modes 198
User-defined Zmacs Major Modes 198
Zmacs Manual 1
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 Notation Conventions 9
Zmacs Midas Mode 157
Built-in Customization Using Zmacs Minor Modes 195
Definition of Zmacs Minor Modes 195
Summary of Zmacs Minor Modes 196
How Zmacs Minor Modes Work 195
Zmacs Mode Line 19
Example 1 of Zmacs Notation Conventions 9

Example 2 of Zmacs Notation Conventions 9
Example 3 of Zmacs Notation Conventions 9
Zmacs Notation Conventions and Examples 9
Zmacs Pl1 Mode 157
What is a Zmacs Region? 86
Zmacs Reverse Incremental Search 99
Description of Zmacs Sentence Delimiters 62, 83
Zmacs Sorting Commands 112
Zmacs String Search 100
Zmacs Teco Mode 157
Zmacs text formatting commands 37
Zmacs Text Mode 156
Definition of a Zmacs Variable 210
Zmacs variables 46
Describing Zmacs Variables 210
Finding Out About Zmacs Variables 210
Listing Zmacs Variables 210
Settable Zmacs Variables 211
How to Specify Zmacs Variable Settings 210
Leaving Zmacs Via the System Menu 40
Zmacs Window Commands 129
Leaving Zmacs with c-Z 40
Entering Zmacs with **ed** 12
Entering Zmacs with SELECT E 12
Entering Zmacs with the **Mouse** 12
Leaving Zmacs with the SELECT Key 40
Entering Zmacs with **zwei:edit-functions** 13
Entering Zmail 40
Hardcopying From Zmail 268
[Hardcopy] Zmail menu item 268
[Map over] Zmail menu item 268
[Move] Zmail menu item 268
zwei:command-store 201
zwei:define-keyboard-macro 201
zwei:defmajor 198
Entering Zmacs with **zwei:edit-functions** 13
zwei:edit-functions function 13
zwei:*file-versions-kept* variable 135
Zwei:*Inhibit-fancy-loop Indentation 162
zwei:*major-mode-translations* Lisp variable 198
zwei:make-macro-command 201
zwei:*send-mail-about-patch* 190
zwei:*set-attribute-update-list* global variable 141, 143
zwei:*temp-file-type-list* variable 135
zwei:*zmacs-comtab* 201

\ \ \
@\ 37
\ Font Editor command 261

@^ 37
c-X ^ Zmacs command 129