The MG Reference Manual

Release MG2A

Sandra J. Loosemore

Contents

Chapter 1

Introduction

MG is a small, fast, and portable Emacs-style text editor intended to be used by people who can't run a real Emacs for one reason or another — as their main editor on smaller machines with limited memory or file space, or as a "quick-start" editor on larger systems, useful for composing short mail messages and the like.

We've made MG compatible with GNU Emacs because that is the "big", full-featured editor that many of us use regularly and are most familiar with. GNU Emacs is the creation of Richard M. Stallman, who was also the author of the original Emacs editor. However, MG is not associated in any way with the GNU project, and the MG authors individually may or may not agree with the opinions expressed by Richard Stallman and the GNU project.

MG is largely public domain. You can use, modify, and redistribute MG as you like. A few modules, however, are copyrighted; specifically, the regular expression code, the VMS termcap routines, and the Amiga support code. Look at the source code for the exact copyright restrictions.

There are several other editors in existence which call themselves MicroEmacs. The original public domain version was written by Dave Conroy and circulated as version 1.6. Derived from this, there is another PD version by Dave Conroy numbered v30; a significantly larger PD version by Daniel Lawrence which is now up to version 3.9; at least one proprietary implementation; an implementation for the Atari ST with an integrated command shell, by Prabhaker Mateti; and probably others that we don't know about.

MG is derived from the v30 MicroEmacs, with key bindings, command names, and general functionality made more compatible with GNU Emacs. Like v30, MG is fairly small and quite robust. We have generally resisted the temptation to overfeaturize. Some features which are large and complex are flagged for conditional compilation.

Many people have contributed their time to developing, improving, and porting MG. Mike Meyer, Mic Kaczmarczik, and Bob Larson deserve particular mention for their efforts.

Questions, suggestions, and offers of help should be addressed to:

```
mg-developers@ucbvax.berkeley.edu    (ARPA)
ucbvax!mg-developers                     (UUCP)
```

1.1 Implementations of MG

MG runs on many different kinds of hardware under many different operating systems. Currently, these include:

- 4.2 and 4.3 BSD Unix (including Ultrix-32)
- System V Unix
- VAX/VMS

- Primos
- OS9/68k
- Amiga
- Atari ST
- MS-DOS

This document describes release MG2A. When we talk of different versions of MG in this manual, the term *version* is used to refer to the different support MG provides for the various machines and operating systems it runs under, not to different releases of MG itself. For example, we might speak of how the VMS version of MG differs from the Unix version.

As mentioned above, some MG commands may not be implemented in all versions; these are noted in the documentation. Some versions of MG also support features (such as mouse handling) that are not described here.

1.2 A Note on Character Sets

MG uses the 128-character ASCII character set, and provides support for 8-bit characters. Whether the particular version of MG that you are running knows about extended character sets depends on whether your terminal and the host operating system know about them. Moreover, since there is no standard 8-bit character set, the same character codes will probably give different glyphs on different systems. Most versions of MG use the DEC multinational character set.

1.3 Notation and Conventions

In this manual, commands and other things that must be typed in literally are indicated in a typewriter font, like `next-line`. Placeholders such as command argument names use an italic font.

The terms *command* and *function* are synonymous. We often speak of a command being bound to a particular *key*, although you may actually have to type more than one character to form a single key. Most commands are bound to keys with *control* and *meta* modifiers.

To type a control character, use the control key on your keyboard like a shift key: hold down the control key while typing the character. In this manual, we will indicate control characters like `C-x` — here, typing the character "x" while holding down the control key.

Some keyboards also have a meta key that works like the control key. (It may be labelled something else; on the Atari ST, for example, the key marked "Alternate" is the meta key.) If your keyboard doesn't have a meta key, don't panic. You can also use the escape key as a meta prefix; first type the escape, and *then* the character. Meta characters will be indicated as `M-x`.

Besides the meta prefix, two other characters are used as prefixes: `C-x` and `C-h`. A few keys have special notation: `SPC` is the space character, `DEL` is the delete or rubout character, `RET` is carriage return, and `ESC` is the escape character. `NUL` is the null character (ASCII 0), which is usually equivalent to either `C-SPC` or `C-@`.

Uppercase and lowercase characters are generally equivalent in command keystrokes.

When you run MG from a shell, command line arguments are interpreted as the names of files you want to *visit*, or edit. Each file is read into a *buffer* in memory. No changes are actually made to the file until you ask it to be written out to disk.

Within MG, the large top part of the screen serves as a *window* into the buffer being edited. Below this is the *mode line*, which displays the name of the buffer. Finally, at the very bottom of the screen, there is a one-line *minibuffer* which is used for displaying messages and answering questions.

MG keeps track of two pointers into each window, the *point* and the *mark*. The *cursor* appears at the point in the current window, and we often speak of moving the cursor rather than of moving the point. The text between the point and the mark is referred to as the *region*.

Some commands deal with *words* and *paragraphs*. Generally, whitespace and punctuation separate words. Lines that are empty or that contain only spaces or tabs separate paragraphs without being part of a paragraph. A non-empty line that starts with a space or tab also begins a new paragraph.

A number of commands are defined as *toggles*. If no prefix argument is supplied, these commands toggle an action. The action is turned on if a negative or zero argument is supplied, and turned on if a positive argument is supplied.

1.4 Getting Started

This document is intended primarily as a reference manual. If you have never used any Emacs-like text editor before, it is strongly suggested that you run the on-line tutorial supplied with the MG distribution, instead of reading this manual.

Do not be put off by the large number of commands described in this manual! It is possible to get by with only a handful of basic commands. Here are the ones that are probably used most frequently:

`C-p` Move the cursor to the previous line

`C-n` Move the cursor to the next line

`C-b` Move the cursor backwards

`C-f` Move the cursor forwards

`C-v` Scroll forwards one screenful

`M-v` Scroll backwards one screenful

`M-<` Go to the beginning of the buffer

`M->` Go to the end of the buffer

`C-a` Go to the beginning of the line

`C-e` Go to the end of the line

`DEL` Delete the previous character

`C-k` Kill (delete) to the end of line

`C-y` Reinsert killed text.

`C-x C-c` Exit MG

`C-x C-s` Save the current buffer

Chapter 2

Using Commands

2.1 Command Arguments

Some commands require arguments. For example, if you want to read a file into a buffer, you must type in the name of the file. In the descriptions of commands in this manual, if arguments are required, they are listed following the command name.

MG prompts for command arguments in the minibuffer. Within the minibuffer, the following characters can be used for editing:

`DEL, C-h` Erase the last character.

`C-x, C-u` Erase the entire input line.

`C-w` Erase to the beginning of the previous word.

`C-q, \` Quote the next character typed.

`RET` Signifies that you have completed typing in the argument.

`C-g` Abort the command in progress.

2.2 Prefix Arguments

All commands accept an optional numeric prefix argument. This is often interpreted as a repetition count. For example, the function `next-line`, if given a prefix argument, will move the cursor forward that many lines; without an argument, it will move the cursor forward one line. A few commands behave differently if given a prefix argument than they do without one, and others ignore the prefix argument entirely.

`digit-argument` `M-0, M-1, M-2, M-3, M-4, M-5, M-6, M-7, M-8, M-9`

`negative-argument` `M-`

One way to specify a command argument is to use the escape key as a meta prefix, and then type one or more digits. A dash may be used for a negative argument.

`universal-argument` `C-u`

Another way to specify a command prefix is to type `C-u`. Typing one `C-u` is equivalent to a prefix argument of 4, typing two gives a value of 16, and so on. In addition, you can type digits following `C-u` to form a numeric prefix argument.

2.3 Aborting

`keyboard-quit` `C-g`

Typing `C-g` cancels any command. It is particularly useful for cancelling a command when MG is prompting for input in the minibuffer.

2.4 Extended Commands

`execute-extended-command` *command* `M-x`

Commands that are not bound to keys can be executed through `execute extended-command`. If a prefix argument is supplied, it is passed to the command being executed.

Chapter 3

Moving the Cursor

The commands described in this chapter move the cursor (sometimes called the point or dot) within the current window. Commands which set the mark are included here as well.

`backward-char` `C-b`

Moves the cursor backward (left) one character. If the cursor is at the left margin, it will be moved to the end of the previous line.

`backward-paragraph` `M-[`

Moves the cursor backwards to the beginning of the current paragraph, or to the beginning of the previous paragraph if the cursor is already at the beginning of a paragraph.

`backward-word` `M-b`

Moves the cursor backwards to the beginning of the current word, or to the beginning of the previous word if the cursor is already at the beginning of a word.

`beginning-of-buffer` `M-<`

Moves the cursor backwards to the beginning of the buffer.

`beginning-of-line` `C-a`

Moves the cursor backwards to the beginning of the current line. This command has no effect if the cursor is already at the beginning of the line.

`end-of-buffer` `M->`

Moves the cursor forwards to the end of the buffer.

`end-of-line` `C-e`

Moves the cursor forwards to the end of the current line. This command has no effect if the cursor is already at the end of the line.

`exchange-point-and-mark` `C-x C-x`

Set the mark at the current cursor position, and move the cursor to the old location of the mark.

`forward-char` `C-f`

Moves the cursor forwards one character. If the cursor is at the end of a line, it will be moved to the first character on the next line.

`forward-paragraph` `M-]`

Moves the cursor forwards to the next paragraph delimiter.

`forward-word` `M-f`

Moves the cursor forwards to the end of the current word, or to the end of the next word if the cursor is already at the end of a word.

`goto-line` *line-number*

Moves the cursor to the beginning of line *line-number* in the buffer.

`next-line` `C-n`

Moves the cursor down one line. The cursor remains in the same column unless it would be past the end of the line, in which case it is moved to the end of the line. At the end of the buffer, `C-n` will create new lines.

`previous-line` `C-p`

Moves the cursor up one line. The cursor remains in the same column unless it would be past the end of the line, in which case it is moved to the end of the line.

`recenter` `C-l`

Redraws the entire screen, scrolling the current window if necessary so that the cursor is near the center. With a positive prefix argument *n*, the window is scrolled so that the cursor is *n* lines from the top. A negative prefix argument puts the cursor that many lines from the bottom of the window.

`redraw-display`

Redraws the entire screen, but never scrolls.

`scroll-down` `M-v`

Scrolls the display down (moving backward through the buffer). Without an argument, it scrolls slightly less than one windowful. A prefix argument scrolls that many lines.

`scroll-one-line-down`

`scroll-one-line-up`

These functions are similar to `scroll-down` and `scroll-up` (respectively), but when invoked without an argument, cause the display to scroll by one line only. These functions are enabled by defining the compile-time option GOSMACS.

`scroll-other-window` `M-C-v`

Scrolls the "other" window forward as for `scroll-up`.

`scroll-up` `C-v`

Scrolls the display up (moving forward through the buffer). Without an an argument, it scrolls slightly less than one windowful. A prefix argument scrolls that many lines.

`set-mark-command` `NUL`

Set the mark at the current cursor position.

`what-cursor-position` `C-x =`

Prints some information in the minibuffer about where the cursor is.

Chapter 4

Text Insertion Commands

The usual way to insert text into a buffer is simply to type the characters. The default binding for all of the printing characters (`self-insert-command`) causes them to be inserted literally at the cursor position.

`insert` *string*

Insert *string* into the current buffer at the cursor position.

`newline` `RET`

Insert a line break into the current buffer at the cursor position, moving the cursor forward to the beginning of the new line.

`newline-and-indent` `C-j`

Insert a line break into the current buffer at the cursor position, then add extra whitespace so that the cursor is aligned in the same column as the first non-whitespace character in the previous line.

`open-line` `C-o`

Inserts a line break into the current buffer at the current position, but does not move the cursor forward.

`quoted-insert` `C-q`

This command acts as a prefix to cancel the normal interpretation of the next keystroke. If `C-q` is followed by one to three octal digits, it is interpreted as the code of the character to insert. Otherwise a single key is read and the character typed is inserted into the buffer instead of interpreted as a command. This is used for inserting literal control characters into a buffer.

`self-insert-command`

This is the default binding for keys representing printable characters. The character is inserted into the buffer at the cursor position, and the cursor moved forward.

Chapter 5

Killing, Deleting, and Moving Text

When text is deleted, it is erased completely. Killing text, on the other hand, moves it into a temporary storage area called the kill buffer. The saved text in the kill buffer is erased when another block of text is killed. Until then, however, you can retrieve text from the kill buffer. This can be used to move or copy blocks of text, as well as to restore accidentally killed text.

`backward-kill-word` `M-DEL`

Kill the text backwards from the cursor position to the beginning of the current word. Typing `M-DEL` several times in succession prepends each killed word to the kill buffer.

`copy-region-as-kill` `M-w`

Copies the text in the region into the kill buffer, without removing it from the current buffer.

`delete-backward-char` `DEL`

Deletes the character to the left of the cursor.

`delete-blank-lines` `C-x C-o`

Deletes all blank lines after the current line, and if the current line is blank, deletes it and all blank lines preceeding it as well.

`delete-char` `C-d`

Deletes the character underneath the cursor.

`delete-horizontal-space` `M-\`

Deletes all spaces and tabs on either side of the cursor.

`just-one-space` `M-SPC`

This is like `delete-horizontal-space`, except it leaves a single space at the cursor position.

`kill-line` `C-k`

If no prefix argument is specified, this function kills text up to the next newline; or if the cursor is at the end of a line, the newline is killed. A prefix argument specifies how many lines to kill. Typing `C-k` several times in succession appends each line to the kill buffer.

`kill-paragraph`

This command kills the entire paragraph containing the cursor. If the cursor is positioned between paragraphs, the next paragraph is killed.

`kill-region` `C-w`

The region (all text between point and mark) is killed.

`kill-word` `M-d`

Text is killed forward from the cursor position to the next end of word. If the cursor is at the end of the word, then the next word is killed. Typing `M-d` several times appends the killed text to the kill buffer.

`yank` `C-y`

Text is copied from the kill buffer into the current buffer at the cursor position. The cursor is moved to the end of the inserted text.

Chapter 6

Searching and Replacing

6.1 Searching

The ordinary search command in MG differs from that in many other editors in that it is incremental: it begins searching as soon as you begin typing the search string, instead of waiting for you to type the entire string.

All of the search commands described in this section are case-insensitive.

`isearch-backward` *pattern*`C-r`

`isearch-forward` *pattern*`C-s`

These commands perform an incremental search backward and forward (respectively) for *pattern*. MG will move the cursor to the place in the buffer that matches as much of the pattern as you have typed so far, as each character is entered.

Within the incremental search, the following characters are interpreted specially:

`DEL` Erase the last character in the search string.

`ESC` Stop searching; exit from incremental search mode, leaving the cursor where the search brought it.

`C-g` If a match has been found, exits from incremental search but leaves the cursor in its original position. If the search has failed, this will just erase the characters which have not been found from the end of the search pattern. In this case, you must type `C-g` again to abort the search.

`C-s` Search forward for the next occurrence of the same pattern.

`C-r` Search backward for the previous occurrence of the same pattern.

`C-q` "Quotes" the next character typed, forcing it to be interpreted as a literal character in the search pattern.

In addition, normal commands such as `C-a` that do not have special meanings within incremental search cause the search to be terminated, and then are executed in the ordinary way.

`search-again`

`search-backward` *pattern*`M-r`

`search-forward` *pattern*`M-s`

These commands perform ordinary, non-incremental searches. `Search-again` uses the same pattern and direction as the previous search.

6.2 Replacing

`query-replace` *pattern replacement*`M-%`

The primary replace command in MG is an interactive query replace. MG searches forward for occurrences of *pattern*, and asks you what to do about each one. The choices are:

`SPC` Replace this match with *replacement*, and go on to the next.

`DEL` Skip to the next match without replacing this one.

`.` Replace this match, and then quit.

`!` Replace all remaining occurrences without asking again.

`ESC` Quit.

By default, `query-replace` adjusts the case of lower-case letters in the replacement string to match that of the particular occurrence of the pattern; for example, replacing “Foo” with “bar” results in “Bar”. Upper case letters in the replacement string are always left uppercase. In addition, supplying a prefix argument will also tell `query-replace` to leave the case of the replacement string as-is.

Note that `query-replace` always performs a case-insensitive search.

6.3 Regular Expressions

Regular expressions provide a means for specifying complex search patterns, instead of just a literal string. The commands in this section are available only if MG is compiled with the REGEX option defined.

Regular expression syntax uses the following rules. Most characters in a regular expression are considered to be *ordinary* characters, and will match themselves and nothing else. The exceptions are the special characters listed below.

`.` Matches any single character except a newline.

`*` A suffix operator that matches zero or more repetitions of the (smallest) preceding regular expression.

`+` A suffix operator that matches one or more repetitions of the (smallest) preceding regular expression.

`?` A suffix operator that matches either zero or one occurence of the (smallest) preceding regular expression.

`[…]` Matches any one character listed in the character set between the square brackets. See examples below.

`^` Matches at the beginning of a line.

`$` Matches at the end of a line.

`\` Except for the situations listed below, acts as a prefix operator which causes the character following to be treated as an ordinary character.

`\|` An infix binary *or* operator. It applies to the two largest surrounding expressions.

`\(…\)` A grouping construct, usually used to specify a larger expression for postfix operators such as `*` or to limit the scope of operands to `\|`.

`\`*`digit* Matches the same text matched by the *digit*th `\(...\)` construct. These are numbered from 1 to 9 in the order that the open-parentheses appear.

`\‘` Matches at the beginning of the buffer.

`\’` Matches at the end of the buffer.

`\b` Matches at the beginning or end of a word.

`\B` Matches anyplace *except* at the beginning or end of a word.

`\<` Matches at the beginning of a word.

`\>` Matches at the end of a word.

`\w` Matches any word-constituent character.

`\W` Matches any character which is *not* a word-constituent.

Some examples may help clarify the rules.

`foo` Matches the literal string `foo`.

`;.*` Matches all strings which begin with a semicolon and continue to the end of a line.

`c[ad]+r` Matches strings of the form `car`, `cdr`, `caar`, `cadr`, and so on.

`[a-z]` Matches any lowercase letter.

`[^a-z]` Matches any character *except* lowercase letters.

`[0-9+-]` Matches a digit or sign.

`\(foo\|bar\)` Matches either the string `foo` or the string `bar`.

`count-matches` *pattern*

`count-non-matches` *pattern*

These commands count the number of lines which do or do not (respectively) match the specified pattern.

`delete-matching-lines` *pattern*

`delete-non-matching-lines` *pattern*

These commands delete all lines which do or do not (respectively) match the specified pattern.

`query-replace-regexp` *pattern replacement*

This is the regular expression version of `query-replace`.

The *replacement* string may be a constant, or it can refer to all or part of the string matched by the *pattern*. `\&` in the replacement string expands into the entire text being replaced, while `\`*n* (where *n* is a number) replaces the *n*th parenthesized expression in *pattern*.

`re-search-again`

`re-search-backward` *pattern*

`re-search-forward` *pattern*

These are the regular expression equivalents of the ordinary non-incremental search commands.

`set-case-fold-search`

This command toggles an internal variable that controls whether the regular expression search and replace commands pay attention to case. By default, regular expression searches are case-insensitive. Ordinary searches are always case-insensitive and are not affected by the setting of this variable.

Chapter 7

Windows

MG initially has only one text window displayed. However, you can have as many windows as will fit on the screen. Each window has its own mode line and must display at least two lines of text. (Note that a MG's "windows" are distinct from the "windows" handled by screen managers such as the X Window System.)

Multiple windows may be used to display different buffers. You can also have the same buffer displayed in more than one window, which is useful if you want to see one part of a file at the same time as you are editing another part.

Although many windows can be displayed at once, only one window is active at any given time. This is the window where the cursor appears.

Some commands refer to the "other" window. This is the window directly below the current window, or the top window if you are in the bottom window.

`delete-other-windows` `C-x 1`

Makes the current window the only window.

`delete-window` `C-x 0`

Deletes the current window, making the "other" window the current window. This command doesn't do anything useful if there is only one window being displayed.

`enlarge-window` `C-^`

Makes the current window larger. Without a prefix argument, the window grows one line; otherwise, the prefix argument specifies how many lines to grow.

`other-window` `C-x o`

Makes the "other" window the current window.

`previous-window`

This is like `other-window`, except that it cycles through the windows in reverse order. This command is available only if MG was compiled with the GOSMACS option defined.

`shrink-window`

Makes the current window smaller. Without a prefix argument, the window loses one line; otherwise, the prefix argument specifies how many lines go away.

`split-window-vertically` `C-x 2`

Split the current window into two windows, both using the same buffer.

Chapter 8

Files and Buffers

Most buffers are used to contain a file being edited. It is also possible to have buffers that are not associated with any file; MG uses these for purposes such as displaying help text, for example. However, since most commands for dealing with files also deal with buffers, we have grouped all of these commands together into one chapter.

8.1 Buffer Manipulation

`insert-buffer` *buffer-name*

Inserts the contents of the named buffer into the current buffer at the cursor location. The cursor moves to the end of the inserted text.

`kill-buffer` *buffer-name*`C-x k`

The named buffer and its contents are deleted. If the buffer has been marked as modified, MG will ask you if you really want to delete it. Note that, contrary to its name, this command *does not* save the buffer contents in the kill buffer.

If a buffer is being displayed in a window when it is deleted, MG will find some other buffer to display in the same window.

`list-buffers` `C-x C-b`

This command writes information about the buffers currently in use to a buffer named `*Buffer List*`. This buffer is then displayed in the "other" window; if there is only one window, this command will split the screen into two windows.

`not-modified` `M-~`

This command makes MG think that the current buffer has not been modified, even if it really has been changed. This affects the behavior of the `kill-buffer` and the buffer-saving commands described below.

MG indicates modified buffers with two stars at the left end of the mode line.

`switch-to-buffer` *buffer-name*`C-x b`

The current window is mapped onto the named buffer. If there isn't already a buffer with that name around, MG will create one.

`switch-to-buffer-other-window` *buffer-name*`C-x 4 b`

This command works like `switch-to-buffer`, except that the "other" window is used. If there is only one window, this command splits the screen into two windows and maps the named buffer onto one of them.

8.2 Reading and Writing Files

`find-file` *file-name* `C-x f`

`find-file-other-window` *file-name* `C-x 4 C-f`

These commands are analagous to `switch-to-buffer` **and** `switch-to-buffer-other-window`**, respectively. The difference is that these commands look for a buffer associated with the named file. If no matching buffer is found, MG will create a new buffer with a name derived from the filename, and attempt to read the file into the buffer. If the named file cannot be opened, the buffer remains empty.**

`insert-file` *file-name* `C-x i`

This command reads in the contents of the named file into the current buffer at the cursor position. The cursor remains in the same place.

`save-buffer` `C-x C-s`

If the current buffer has been modified, it is saved. Buffers that are not associated with files cannot be written out with this command.

`save-buffers-kill-emacs` `C-x C-c`

This command is used to leave MG and return control to the shell or other program that was used to start MG. If there are modified buffers, MG will ask you if you want to save them before exiting.

`save-some-buffers` `C-x s`

MG will ask you if you want to save modified buffers that are associated with files.

`write-file` *file-name* `C-x C-w`

The current buffer is written out using the file name supplied. This is useful for saving buffers that are not associated with files, or for writing out a file with a different name than what was used to read it in.

8.3 Backup Files

MG provides a way to save a copy of the original version of files which have been modified and then written out again. The backup copy reflects the state of the file as it existed the first time it was read into MG. The name used for the backup file varies, depending on the operating system.

This feature is disabled if MG is compiled with NO_BACKUP defined.

`make-backup-files`

This command is a toggle which controls the state of an internal variable that determines whether MG creates backup files.

8.4 Changing the Directory

The commands in this section are disabled by defining NO_DIR.

`cd` *directory-name*

This command changes MG's notion of the "current" directory or pathname. This is used to supply defaults for functions that read or write files.

The syntax for *directory-name* is obviously specific to the particular operating system MG is running on.

`pwd`

Display what MG thinks is the current directory.

Chapter 9

Modes

Modes are used to locally alter the bindings of keys on a buffer-by-buffer basis. MG is normally in fundamental mode, and these are the bindings that are listed with the command descriptions in this manual. Modes define additional keymaps that are searched for bindings before the fundamental mode bindings are examined; see the section on key binding below for more details on how this works.

`set-default-mode` *mode-name*

Normally, when MG visits a file, it puts the associated buffer into fundamental mode. Using the `set-default-mode` command, you can specify that MG should default to use some other mode on all subsequent buffers that are created. This command is a toggle. With no prefix argument, if the named mode is not already on the list of default modes, then it will be added to the list; otherwise, it is removed from the list.

9.1 No Tab Mode

In notab mode, tabs are expanded into spaces instead of inserted literally into the buffer. Literal tab characters are displayed as `^I` (much like other control characters). These commands are available if MG is compiled with the symbol NOTAB defined. (This mode is mainly for use on systems such as PRIMOS that do not treat tab as a series of spaces.)

`no-tab-mode`

This command is a toggle to control whether notab mode is in effect.

`space-to-tabstop`

Insert enough spaces to move the cursor to the next tab stop. In notab mode, this function is bound to `C-i`.

9.2 Overwrite Mode

Normally, when characters are inserted into the buffer, they are spliced into the existing text. In overwrite mode, inserting a character causes the character already at the cursor position to be replaced. This is useful for editing pictures, tables, and the like.

`overwrite-mode`

This command is a toggle which controls whether overwrite mode is in effect.

9.3 Auto Fill

Fill mode causes newlines to be added automatically at word breaks when text is added at the end of a line, extending past the right margin. Auto fill is useful for editing text and documentation files.

`auto-fill-mode`
This command is a toggle which controls whether fill mode is in effect.
`insert-with-wrap`
This command works like `self-insert`, except that it checks to see if the cursor has passed the right margin. If so, it fills the line by inserting a line break between words. This command is bound to `SPC` in fill mode.
`fill-paragraph` `M-q`
Fill the paragraph containing the cursor.
`set-fill-column` `C-x f`
Without a prefix argument, this command sets the right margin at the current cursor column. If a prefix argument is supplied, it is used instead as the line width.

9.4 Auto Indent

Indent mode binds `RET` to `newline-and-indent`, so that each new line is indented to the same level as the preceeding line. This mode is useful for editing code.
`auto-indent-mode`
This command is a toggle which controls whether auto-indent mode is in effect.

9.5 Blink

Blink mode makes it easier to match parentheses, brackets, and other paired delimiters. When the closing delimiter is typed, the cursor moves momentarily to the matching opening delimiter (if it is on the screen), or displays the line containing the matching delimiter on the echo line. This is useful for editing Lisp or C code, or for preparing input files for text processors such as LaTeX that use paired delimiters.
`blink-matching-paren`
This command is a toggle which controls whether blink mode is in effect.
`blink-matching-paren-hack`
This function behaves like `self-insert`, except that it finds the matching delimiter as described above. In blink mode, this function is bound to `)`, which flashes the matching `(`. This function also knows about the pairs `{}`, `[]`, and <>. All other characters match with themselves.

9.6 Dired Mode

“Dired” is an abbreviation for “directory editor”, and it provides a way to browse through the contents of a directory from with MG. Dired puts a directory listing into a buffer; you can use normal editing commands to move around the buffer, and a special group of commands to manipulate the files. For example, there are commands to delete and rename files, and to read a file into an MG buffer.

Since dired mode rebinds many keys, a table may be helpful:

```
C-d         dired-flag-file-deleted
SPC         next-line
```

```
c           dired-copy-file
d           dired-flag-file-deleted
e           dired-find-file
f           dired-find-file
n           next-line
o           dired-find-file-other-window
p           previous-line
r           dired-renamefile
u           dired-unflag
x           dired-do-deletions
DEL        dired-backup-unflag
```

The commands in this section are disabled by defining NO_DIRED.

`dired` *directory-name*`C-x d`

Creates a dired buffer for the given directory name, and displays it in the current window. The files in the directory are listed, usually along with information about the file such as its size and timestamp. The exact format of the information is system-specific.

`dired-backup-unflag`

This function removes the deletion flag from the file listed on the previous line of the dired buffer.

`dired-copy-file` *new-name*

Copy the file listed on the current line of the dired buffer.

`dired-do-deletions`

Deletes the files that have been flagged for deletion.

`dired-find-file`

`dired-find-file-other-window`

These function works like `find-file` and `find-file-other-window`, except that the filename is taken from the current line in the dired buffer.

`dired-flag-file-deleted`

Flag the file listed on the current line for deletion. This is indicated in the buffer by putting a “D” at the left margin. No files are not actually deleted until the function `dired-do-deletions` is executed.

`dired-other-window` *directory-name*

This function works just like `dired`, except that it puts the dired buffer in the “other” window.

`dired-rename-file` *new-name*

Renames the file listed on the current line of the dired buffer. Note that the dired buffer is not updated to reflect the change.

`dired-unflag`

Remove the deletion flag for the file on the current line.

Chapter 10

Miscellaneous

10.1 Help

Most of the commands in this section write useful information to the `*help*` buffer, which is then displayed in the "other" window.

These commands can be disabled at compile-time by defining NO_HELP.

`apropos` *topic* `C-h a`

This command lists all functions whose names contain a string matching *topic* in the `*help*` buffer.

`describe-bindings` `C-h b`

Information about the key bindings in effect in the current buffer is listed in the `*help*` buffer.

`describe-key-briefly` *key* `C-h c`

Information about the binding of *key* is printed in the minibuffer.

`help-help` *option* `C-h C-h`

This command lists all of the help options available and prompts for which one to run. Currently, these include only `a` to run `apropos`, `b` to run `describe-bindings`, and `c` to run `describe-key-briefly`.

10.2 Keyboard Macros

A keyboard macro is a saved set of commands from the keyboard that can be reexecuted later on. There can only be one keyboard macro defined at any one time.

The commands in this section are available unless they have been disabled by defining NO_MACRO.

`call-last-kbd-macro` `C-x e`

Execute the saved keyboard macro. A prefix argument can be used to specify a repetition count.

`end-kbd-macro` `C-x )`

`start-kbd-macro` `C-x (`

These functions are used to define a keyboard macro. All keys entered after `start-kbd-macro` is executed, up to a `end-kbd-macro`, are remembered as they are executed. You can then reexecute the same sequence of operations using `call-last-kbd-macro`.

10.3 Changing Case

MG provides a number of functions for changing the case of text.

`capitalize-word` `M-c`

`downcase-region` `C-x C-l`
`downcase-word` `M-l`
`upcase-region` `C-x C-u`
`upcase-word` `M-u`
All of these commands do the obvious.

10.4 Odds and Ends

This section describes miscellaneous commands that don't fit into any particular category.
`emacs-version`
Prints information about the version of MG you are running in the minibuffer.
`meta-key-mode`
If the particular version of MG you are running supports a meta key, this function can be used to determine whether MG actually pays attention to it or not. If no prefix argument is supplied, the internal variable that controls the use of the meta key is toggled; a positive value enables the meta key, while a negative value disables it.
`prefix-region`
`set-prefix-string` *string*
`Prefix-region` is used to prefix each line of the region with a string. This is useful for indenting quoted text, making block comments, and the like. The function `set-prefix-string` can be used to set the string used as the prefix.
`suspend-emacs` `C-z`
This command temporarily suspends MG so that you can run other programs, and later resume editing. The exact behavior depends on which operating system you are running MG under. Typically, MG will either spawn a new shell as a subprocess, or return you to the parent process.
`transpose-chars` `C-t`
This command transposes the previous two characters.

Chapter 11

Customization

MG provides a limited support for customization. However, unlike “real” Emacs, there is no extension language for interpretively defining new functions.

11.1 Key Bindings

MG allows keys to be rebound locally or globally. To understand the difference between the two, some discussion on how modes are implemented is necessary.

An internal data structure called a keymap is used to look up the function that is bound to a particular key. The keymap for fundamental mode contains all of the default bindings which are listed with the command descriptions in this manual. Modes define additional keymaps that are searched for a binding before the fundamental mode keymap is examined. Keymaps have the same name as the mode they are associated with.

MG does not provide commands for defining new modes, but you can alter the keymaps for existing modes.

`define-key` *keymap-name key command*

This command can be used to modify the keymap for the named mode.

`global-set-key` *key command*

`global-unset-key` *key*

These commands modify the keymap for fundamental mode. Bindings established by `global-set-key` will be inherited by all other modes, as long as they do not establish local rebindings of the same key.

`local-set-key` *key command*

`local-unset-key` *key*

These commands modify the keymap currently in effect.

11.2 Startup Files

Although MG does not include a general-purpose extension language, it does provide a way to read and evaluate commands using a somewhat different syntax than that used for executing extended commands. This is typically used in a startup file to modify key bindings.

A startup file consists of one or more expressions. Each expression must appear on a separate line in the file; there may not be more than one expression per line, nor may expressions span across line breaks. Whitespace (spaces and tabs) separate the tokens in an expression. For historical reasons, parentheses are also considered to be whitespace in this context. A semicolon acts as a comment character, causing the rest of the line to be discarded.

An expression consists of a function name, an optional prefix argument (given as an integer constant), and arguments to be passed to the function. If an argument includes literal whitespace

or nonprintable characters (for example, as in a keystroke argument to one of the key binding functions described in the previous section), it must be supplied as a string constant enclosed in double quotes.

Within string constants, the following backslash escapes are available to specify nonprintable characters:

`\t, \T` Tab

`\n, \N` Newline

`\r, \R` Carriage return

`\e, \E` Escape (Meta prefix)

`\^` Control prefix

`\`*n* Specifies a character by its ASCII code, where *n* may consist of from one to three octal digits

`\f`*n*`, \F`*n* Specifies the keycode for the *n*th function key. *N* may consist of one or two decimal digits.

The following commands which deal with evaluation of expressions are disabled by defining the compile-time option NO_STARTUP. See the implementation notes for your particular version of MG for information on how it handles startup files.

`eval-current-buffer`
Evaluate the expressions in the current buffer.

`eval-expression` *expression*
Evaluate the expression supplied.

`load` *file-name*
Read in the specified file and evaluate its contents.

[Fundamental Mode Key Bindings]

NUL	set-mark-command
C-a	beginning-of-line
C-b	backward-char
C-d	delete-char
C-e	end-of-line
C-f	forward-char
C-g	keyboard-quit
C-h	help
TAB	self-insert-command
C-j	newline-and-indent
C-k	kill-line
C-l	recenter

RET	newline
C-n	next-line
C-o	open-line
C-p	previous-line
C-q	quoted-insert
C-r	isearch-backward
C-s	isearch-forward
C-t	transpose-chars
C-u	universal-argument
C-v	scroll-up
C-w	kill-region
C-x	c-x prefix
C-y	yank
C-z	suspend-emacs

```
ESC             meta prefix
SPC .. ~    self-insert-
command
DEL             delete-
backward-char

C-h C-g     keyboard-quit
C-h C-h     help-help
C-h a        apropos
C-h b        describe-
bindings
C-h c        describe-key-
briefly

C-x C-b     list-buffers
C-x C-c     save-buffers-
kill-emacs
C-x C-f     find-file
```

C-x C-g keyboard-quit
C-x C-l downcase-region
C-x C-o delete-blank-lines
C-x C-s save-buffer
C-x C-u upcase-region
C-x C-w write-file
C-x C-x exchange-point-and-mark
C-x (start-kbd-macro
C-x) end-kbd-macro
C-x 0 delete-window
C-x 1 delete-other-windows
C-x 2 split-window-vertically
C-x 4 c-x 4 prefix

C-x = what-cursor-position
C-x ^ enlarge-window
C-x b switch-to-buffer
C-x d dired
C-x e call-last-kbd-macro
C-x f set-fill-column
C-x i insert-file
C-x k kill-buffer
C-x o other-window
C-x s save-some-buffers
C-x 4 C-f find-file-other-window
C-x 4 C-g keyboard-quit

C-x 4 b switch-to-buffer-other-window
C-x 4 f find-file-other-window

M-C-g keyboard-quit
M-C-v scroll-other-window
M-SPC just-one-space
M-% query-replace
M-- negative-argument
M-0 digit-argument
M-1 digit-argument
M-2 digit-argument
M-3 digit-argument
M-4 digit-argument
M-5 digit-argument

```
M-6             digit-argument
M-7             digit-argument
M-8             digit-argument
M-9             digit-argument
M-<             beginning-of-
buffer
M->             end-of-buffer
M-[             backward-
paragraph
M-\             delete-
horizontal-space
M-]             forward-
paragraph
M-b             backward-word
M-c             capitalize-
word
M-d             kill-word
M-f             forward-word
```

```
M-l             downcase-word
M-q             fill-paragraph
M-r             search-
backward
M-s             search-forward
M-u             upcase-word
M-v             scroll-down
M-w             copy-region-
as-kill
M-x             execute-
extended-command
M-~             not-modified
M-DEL          backward-kill-
word
```