MicroEMACS:
Reasonable Editing
on
Tiny Computers

by
David G. Conroy
Digital Equipment Corporation
146 Main St.
Maynard, MA 01754

revised by
Mark Alexander
marka@pobox.com
formerly at:
Digital Research, Inc.
70 Garden Court
Box DRI
Monterey, CA 93942

Feb 13, 2018

Contents

1 Introduction

This document describes MicroEMACS, a public-domain[1] display editor, loosely based on EMACS, that runs on little computers. It is intended as a reference manual for users already familiar with EMACS.

We call it Micro *EMACS* to emphasize the fact that most of the commands are very similar to, if not identical to, fundamental mode EMACS commands (actually, it implements a set of rebindings used by a group of users at Digital Equipment Corporation[2]).

We call it *Micro* EMACS to emphasize the fact that it is but a shadow of full EMACS. No attempt has been made to make MicroEMACS wildly customizable (without writing code), or to have extensive online documentation. All of this sophistication was thrown away right at the start, because it was important that MicroEMACS run on little computers. In all fairness, is should be stated here and now that the most popular small computer these days is the MicroVAX[3]!

1.1 History

MicroEMACS is loosely based on the EMACS display editor written by Richard Stallman at MIT. The MicroEMACS described by this document is Conroy's version 30 of February 1986, distributed to USENET mod.sources. Since then it has undergone a fair number of bug fixes and performance improvements, and a few minor enhancements.

This version of MicroEMACS is not to be confused with other popular versions that are derived from Conroy's November 1985 release. These other versions of MicroEMACS have version numbers like 3.8 or 3.9, are considerably larger and feature-rich, are copyrighted, and are now being maintained by Daniel Lawrence.

I (Mark Alexander) converted this document from TeX to Scribe to FinalWord 1.15 to Borland Sprint, and finally to Pandoc/Latex, since Sprint is no longer available and doesn't run on Linux. I added new material to bring the document up to date with the program itself.

1.1.1 2018 Update

I have lost the original source of Conroy's version of MicroEMACS (including the TeX version of this document), and I cannot find it on Google's USENET archive. I have also lost my DOS and OS/2 source code, though the Windows code (in the `nt` subdirectory) still exists. This document contains many reference to historical machines and operating systems, but I have kept them for historical interest. I maintain only the Linux version now.

In the 80s, MicroEMACS was small enough to run easily from a floppy disk, but the amount of text that could be edited was limited by the very small amount of available RAM (640kb on PCs running MS-DOS). Nowadays this limit is effectly non-existent, give the huge amount of memory found in modern computers, but MicroEMACS is still small enough to be run from a floppy disk (if one could be found); its code size is about the same as `/bin/ls` on a current 64-bit Linux distribution.

2 Some Basic Concepts

This section describes a number of basic concepts, which will be used again and again in the following descriptions. Most of it should be familiar to EMACS users. New users are advised not to get stuck reading this section, because most of the information will become clear the first time you play with the editor on a computer.

2.1 Screen Layout

MicroEMACS divides the screen into two areas. The very last line on the screen, which is used to display messages and ask one line questions, is called the *echo line* . Typing in the echo line is like typing commands to the operating system. More details later in the section **The Echo Line**.

The remainder of the screen is filled with text *windows* . When MicroEMACS is first started, there will be one or two text windows, depending on how many files you specify on the invocation line. If there is only one window, it fills the

[1] The source is now copylefted with the GPL.
[2] Further modified at DRI to more closely resemble, but not completely emulate, MINCE..
[3] This appears to be a reference by Conroy to DEC's attempt to ignore the IBM PC.

screen. If there are two windows, each occupies half the screen. Windows can be created, deleted, and adjusted in size using MicroEMACS commands.

2.2 Mode Lines

The last line of a window (the line, usually in reverse video, that starts with `MicroEMACS`) is the *mode line* for the window. It contains information about what is being displayed in the window. Mode lines separate windows from each other and from the echo line.

Three important bits of information are displayed in the mode line. These are the buffer *mode* (if any) in parentheses, the *buffer name* , the *file name* , and the *buffer changed flag* .

A window is always displaying a buffer. The name of the buffer being displayed in the window is shown in the mode line, right after the `MicroEMACS` and the buffer mode.

A buffer may have a file associated with it. Usually this is the last file read into or written from the buffer. If a file is associated with the buffer, the name of the file is displayed in the mode line, prefixed by `File:`. On operating systems in which file names are case insensitive (for example, VMS) the file name will always be in lower case.

MicroEMACS knows if a buffer has been changed since it was last written to its associated file. It informs you of this fact by displaying a `*` in the mode line immediately to the left of the `MicroEMACS` string.

2.3 Keyboard Conventions

Internally, MicroEMACS represents keyboard characters in a special 11 bit code. This code consists of 8 bits of character data (ASCII plus whatever characters your terminal puts in the right side graphics set), and 3 flags, the *CONTROL* flag, the *META* flag, and the *CTLX* flag. Combinations of the three flags are possible, with the exception that META and CTLX cannot be set at the same time.

This manual represents 11 bit characters in a standard way. If you have a character `A`, then that character with the *CONTROL* bit set is written as **C-A**. That character with the *META* bit set is written as **M-A**. That character with the *CTLX* bit set is written as **C-X A**. If both the META and CONTROL bits are set, it is written as **M-C-A**. If both the CTLX and CONTROL bits are set, it is written as **C-X C-A**.

Terminals (usually) cannot deal directly deal with the 11 bit character set. Characters with the flag bits set are entered by prefixing the desired character with another character, as follows:

- To get a character with the *CONTROL* bit set, hold down the `CTRL` key while entering the character. An alternative method is to prefix the character with `Control-^` (the word `Control` here, and in the next few examples, means the key marked `Control` or `CTRL` on the ASCII keyboard).
- To get a character with the *META* bit set, prefix the character with `Control-[`, `Escape`, or `ESC`. An alternative method that works on FlexOS is to hold down the `ALT` key while entering the character. This method also works on PC-DOS, but only with `ALT-A` through `ALT-Z`.
- To get a character with the *CTLX* bit set, prefix the character with `Control-X`.
- To get a character with the *CONTROL* and *META* bits set, prefix the character with `Control-\`. An alternative method is to prefix the character with `ESC`, then hold down `CTRL` while entering the character.
- Those of you who understand ASCII are probably asking what you get if you prefix a character with `Control-]`. This character is not a prefix; it is reserved for future use as an "exit from recursive editing level" command.

CONTROL characters are very important, so there is a shorthand way of typing them, as follows:

- Characters between `Control-@` (`Control-Space` on some keyboards) and `Control-Z` are mapped to the characters `@` through `Z` with the *CONTROL* bit set.
- Characters between `Control-a` and `Control-z` are mapped to the characters `A` through `Z` with the *CONTROL* bit set.

This shorthand means that that **M-C-V**, for example, can be entered either as `ESC` followed by `Control-V`, or as `Control-\` followed by `V`. As another example, **C-X C-B** is entered as `Control-X` followed by `Control-B`.

2.3.1 The PC Keyboard

MicroEMACS understands all of the cursor keypad and function keys on PC compatible machines, and the equivalent keys on other machines that run FlexOS (such as the VME/10). These keys include the function keys (F1-F10), shifted function keys (S-F1 to S-F10), cursor keys (Up, Down, Left, Right, PgUp, PgDn, Home, End), Delete, and Insert. These keys may be combined with the ALT and CTRL keys.

On PCs, you can use the `ALT` key as the META flag, instead of the `ESC` key. For example, the **M-L** command (lower-word) can be executed in either of two ways:

- Hit `ESC` followed by `L`.
- Hold down `ALT` and hit `L`.

On PC-DOS and Linux, the `ALT` key only works with alphabetic characters (`A` through `Z`). Using `ALT` and `CTRL` together is also not supported.

On FlexOS, the `ALT` key works with both alpha and non-alpha characters, and simultaneously with the `CTRL` key. Thus, the command **M-C-V** (display-version) can be executed by holding down `ALT` and `CTRL`, while pressing `V` (not an easy task, in this writer's opinion).

On the VME/10, some of the cursor keys are not available, due to the inherent physical limitations of the keyboard itself.

See the **Wall Chart** section of this manual for the predefined bindings of some the special PC keys.

2.3.2 The Zenith Z-19/29 Keyboard

MicroEMACS treats the numeric keypad on the Z-19/29 similarly to the EDT/VMS customizations in common use at DRI. It does this by putting the terminal in "alternate keypad mode", so that the keypad produces escape sequences instead of numbers. The arrow keys (2, 4, 6 and 8) move one line or one character at a time; when shifted they move by pages or word. When preceded by the F6 key (BLUE on Z-19), the arrow keys move to the beginning or end of the current line or buffer. The 5 key is the same as **C-S** (forw-search), and shift-5 is search-again. See the **Wall Chart** section of this manual for further key bindings for the Z-19/29.

Not all of the key bindings that allow the shifted arrows to work like EDT are built into MicroEMACS. This is because they conflict with the traditional definitions of **M-B** and **M-C**. Therefore, these rebindings are placed in a profile called **edt.pro**. If you want MicroEMACS to read these bindings when it starts up, and you don't care about losing the old meanings of **M-B** and **M-C**, then copy **edt.pro** to the default profile name, as described below in the **Profiles** section.

2.3.3 The VT-100 Keyboard

MicroEMACS understands the arrow keys and the keypad on VT-100 style keyboards. Only the PF1 through PF4 keys on the keypad have any special meaning; the digits, the `-` key, and the `,` key are treated just like the non keypad versions of those characters. The `Enter` key is treated exactly like the `Return` key.

The arrow keys may be used for moving around. The left arrow is the same as **C-B**, the right arrow is the same as **C-F**, the up arrow is the same as **C-P**, and the down arrow is the same as **C-N**.

The four function keys are command keys. The `PF1` key is the same as **M-X** (the `PF1` key is where the `Gold` key normally resides on DEC products, so it is a good choice for the key that asks for extended commands). The `PF2` key is the same as **C-Q** (quote character). The `PF3` key is the same as **C-S** (search forward). The `PF4` key is the same as **C-R** (search reverse). These assignments have not been proven optimal, so they may get changed in the future.

2.3.4 The LK201 Keyboard

The escape key, used in all *META* commands, is very poorly placed on the LK201 keyboard. To make typing a bit easier, the grave accent is also considered to be a *META* prefix on the LK201; the grave accent is right where you expect the escape key to be located. A grave accent must be quoted to exter it into the text.

The arrow keys and all of the keys on the keypad work just like they do on the VT-100 keyboard.

The keys immediately above the arrow keys try to do what their name implies. `Next Screen` is the same as **C-V**. `Prev Screen` is the same as **M-V**. `Select` is the same as **C-@** (set mark). `Remove` is the same as **C-W** (kill region). `Insert here` is the same as **C-Y** (yank from killbuffer). `Find` is the same as **C-S** (search forward).

The `F6`, `F7`, `F8`, `F9`, `F10`, and `F14` keys are unused. The `F11` is escape, which is a *META* prefix key. The `F12` key is backspace. The `F13` key is linefeed.

The `Help` key is unused. The `Do` key is the same as **C-X E** (execute keyboard macro).

The `F17` key is the same as **C-X P** (previous window). The `F18` key is the same as **C-X N** (next window). The `F19` key is the same as **C-X Z** (grow window). The `F20` key is the same as **C-X C-Z** (shrink window).

2.4 Key Bindings

Normally when you type a key, MicroEMACS attempts to interprets the key as a command. Internally, MicroEMACS keeps a table of each possible key, and the command assocated with that key. This association is called the "key binding."
Even text keys, such as the letters, numbers, and punctuation, are "bound" to a command called "ins-self" that just inserts the key that invoked it into the text.

Not all commands are bound to a particular key. These commands can still be entered by using the (**M-X**) (`ESC X`) "extended-command" prefix, or by binding them to a key with the "bind-to-key" command. See the section **Key Binding Commands** below for more information.

In subsequent sections, the command descriptions will give not only the key that the command is normally bound to, but also the command name, which is useful when you want to change the key bindings.

2.5 The Echo Line

The echo line has two purposes; displaying messages and asking one line questions.

Two types of messages are displayed in the echo line. Informational messages tell you something useful, but do not imply in any way that something is wrong. These messages are always enclosed in square brackets ([and]). Error messages indicate that something has prevented a command from being executed. They are never enclosed in square brackets, and will be accompanied by a beep from the terminal's bell if the sound of the bell is not too obnoxious.

The echo line is cleared by the next keystroke.

The echo line is also used for asking and answering questions. After the prompt, you can type any characters you like. The reply is always terminated by a `Return`. Before you commit to the reply, you can edit the reply using the following characters:

- `Backspace` or `Rubout`: delete the character to the left of the cursor.
- `Control-A`: move the cursor to the beginning of the line.
- `Control-B`: move the cursor one character to the left.
- `Control-D`: delete the character under the cursor.
- `Control-E`: move the cursor to end of the line.
- `Control-F`: move the cursor one character to the right.
- `Control-K`: delete from the cursor to the end of the line.
- `Control-Q`: enter the next character literally into the line (useful for entering control characters).
- `Control-U`: delete the entire line.

You can also abort the command in progress by typing `Control-G`. Command processors are designed to ask all questions before doing nasty things, so that you will never do damage by aborting a command.

The echo line supports autocompletion. If you are entering a command name, a filename, or a buffer name, you can use the `Space` or `Tab` keys to tell MicroEMACS to complete as much of the entry as possible, based on what you have entered so far. If you are entering a filename, pressing `Control-S` will fill in the directory part of the current

buffer's filename, making it easier to find a file in the same directory. Pressing the `?` or `Control-D` keys will open a new temporary window containing the possible list of choices.

If you are entering a search string, pressing `Control-S` will fill in the previous search string.

2.6 Command Arguments

All commands can be given a numeric argument. Most commands use this argument as a repeat count. Commands can tell if an argument has been supplied, and will use a default argument if no user argument is present. Usually, the default argument is `1`.

A **C-U** preceding a command always introduces an argument.

If a numeric argument is just a string of **C-U** characters, then the value of the argument is 4^(number of **C-U** characters), where "^" means "to the power." Therefore **C-U** is 4, **C-U C-U** is 16, **C-U C-U C-U** is 256, and so on. Some commands care if an argument is present, but don't look at its value; the **C-U** character is a good way to say "here is an argument".

If there is a signed decimal number after the string of **C-U** characters it specifies the value of the argument. In this case the number of **C-U** characters does not matter. An argument of 10 can be represented by **C-U 10**, **C-U C-U 10**, or even **C-U C-U C-U C-U 10**.

2.7 Dot, Mark, and the Region

Most editing commands act on the character or characters surrounding the current location in the current buffer. The thing that marks the current location is always called *dot* . The dot always points between two characters. This isn't the way that the cursor works on most display terminals, so, by convention, dot is always immediately to the left of the character on which the hardware cursor is placed.

There is also a second remembered position in the buffer, called the *mark* . There are commands that set the value of mark.

Some commands act on a block of text called the *region* . The region is all of the characters between dot and mark. The relative positions of dot and mark do not matter.

3 Starting

Start editing by typing the command

`pe [-b] [-d] [-x] [-z] [-p` *`profile`*`] `*`filename1 filename ...`*

(The name `pe` was chosen to avoid confusion with other versions of MicroEMACS. Think of it as an abbreviation of "puny emacs".)

The options -b, -x, -z, and -p are optional. They have the following meanings:

-b Tells MicroEMACS to make backups of the files you edit. It will create a backup file the first time you save a file that has been changed. It makes the backup by renaming the original file before it writes out the new version. If you have defined an XBACKUP environment variable, MicroEMACS treats it as a directory name and moves the original file into that directory. If you haven't defined XBACKUP, MicroEMACS renames the original file in one of two ways:

- On UNIX-like operating systems, such as Linux,, it appends a "~" character to the filename.
- On PC-DOS and FlexOS, it changes the extension to ".BAK".

This option has no effect on VMS, because VMS supports multiple file versions.

-d Tells MicroEMACS to pass the -d flag to cscope, which tells it to not update its cross-reference file (`cscope.out`). This is useful when using a cross-reference file generated by another program, such as starscope.

-x Tells MicroEMACS to use the XON/XOFF protocol for starting and stopping transmission to your terminal. This is necessary if you are using a DEC terminal, or any terminal that might lose

characters during lengthy operations, such as multi-line scrolls. This option is probably not necessary for Zenith Z-29 terminals, or if your computer uses a memory-mapped display. Unfortunately, this option prevents you from using Control-S and Control-Q while editing. You can get around this problem by reassigning the functions that normally are invoked with these keys to other keys. These key assignments could be placed in your startup profile (see the **Profiles** section below).

-z Tells MicroEMACS to append a Control-Z to output files, and to treat Control-Z as an end-of-file character when reading text files or profiles. This option is provided for compatibility with older editors and language processors. It has no effect on VMS or UNIX.

-p *profile* Tells MicroEMACS to read the specified *profile* at startup, instead of the default profile. See the **Profiles** section for more information about profiles.

-r Tells MicroEMACS to mark all buffers as read-only by default. This is useful in preventing unwanted changes being made when browsing files. A buffer can subsequently be made read-write with the **toggle-readonly** command.

The filenames *filename1*, *filename2*, etc., are the names of the files you want to edit. MicroEMACS will load the specified files into separate buffers, and you can start editing them.

MicroEMACS will create up to two split-screen windows to view the first two files you specify. If you specify only one file, MicroEMACS will create one full-screen window. The commands for manipulating windows are described in the **Window Management** section below.

Any changes you make to a file while it is in a buffer will not affect the original file until you tell MicroEMACS to save the file.

You can omit the filenames when you start. MicroEMACS will present you with an empty buffer called "main", which has no associated filename. You can give this buffer a filename, or read a file into it, or you can read files into their own separate buffers. The commands for reading and saving files are described in the **Files** section below.

After MicroEMACS reads in the files you specify (if any), but before it accepts any keyboard entries, it executes the commands in the startup profile (either the one you specified with the -p option, or the default profile).

4 Quitting

When you finish editing, you have to quit. MicroEMACS never writes out a file because it thinks that this is the right thing to do. However, it will attempt to protect you from forgetting to do so.

C-X C-C **quit**

This is the basic quit command. MicroEMACS exits, returning control to the shell. If there are any changed buffers that have not been written out, it will ask for permission to quit. Supplying an argument to **C-X C-C** makes the command quit unconditionally, without asking for confirmation. The value of the argument is not important. On PCs, this function is also bound to `F4`.

C-C **spawn-cli**

This command suspends the execution of MicroEMACS, and runs a command interpreter in a subjob. When the subjob terminates, the screen is cleared and repainted.

Subjobs are implemented in FlexOS, PC-DOS VMS, and UNIX-like operating systems, such as Linux. Users of CP/M are out of luck.

Exit the command interpreter and return to MicroEMACS with one of the following:

- On VMS enter the **logout** command.
- If you are using the c-shell or bash on Unix-like operating systems, enter the **fg** command.
- On all other systems enter the **exit** command.

C-X C **jeff-exit**

This is a slightly more elaborate quit command. If the current buffer has been changed, **C-X C** saves the contents of the buffer in the associated file (it gets an error if there is no associated file). If the current buffer has not been changed, **C-X C** acts like **C-C**.

5 Moving Around

C-@,C-space,M-space set-mark

Set the value of the mark to be equal to the dot. Use this command to mark the beginning of a region of text to be operated on by a region command such as **C-W**. You can also use this command to mark current location so it can be returned to later by **C-X C-X**. This command also pushes the new value of the mark onto a 16-entry ring of marks for the current window.

If you precede this command with an argument (i.e., precede it with a **C-U**), it sets the dot to the most recent mark that was stored in the ring, and moves that mark to the end of the ring. It also sets the mark to the next most recent mark in the ring, if any. By repeatedly using this command with an argument, you can cycle through the entire ring of marks, returning to those marked locations in sequence.

C-X C-X swap-dot-and-mark

Exchange the positions of the dot and the mark. This is useful for switching back and forth between two points in a file.

C-A goto-bol

Move to the beginning of the current line. Any argument is ignored. Always succeeds. On PCs, this function is bound to the `Home` key.

C-B back-char

Move backwards character by character. The number of characters to move is specified by the argument. If no argument is given it moves backwards by 1 character. A newline counts as a single character. Fails if executed at the beginning of the buffer. On PCs, this function is bound to the `Left` arrow key.

C-E goto-eol

Move to the end of the current line. Any argument is ignored. Always succeeds. On PCs, this function is bound to the `End` key.

C-F forw-char

Move forwards character by character. The number of characters to move is specified by the argument. If no argument is given it moves forwards by 1 character. A newline counts as a single character. Fails if executed at the end of the buffer. On PCs, this function is bound to the `Right` arrow key.

C-N forw-line

Move forward by lines. Attempt to preserve the current horizontal position. The number of lines to move is specified by the argument. If no argument is given it moves by 1 line. Fails if executed at the end of the buffer. On PCs, this function is bound to the `Down` arrow key.

C-P back-line

Move backwards by lines. Attempt to preserve the current horizontal position. The number of lines to move is specified by the argument. If no argument is given it moves by 1 line. Fails if executed at the beginning of the buffer. On PCs, this function is bound to the `Up` arrow key.

C-V forw-page

Move forward by pages. If an argument is given, it specifies the number of pages to move. If no argument is given, 1 is assumed. A page is a group of lines about 20% smaller than a window. If

possible, dot is kept where it is; otherwise it it moved to the middle of the new page. On PCs, this function is bound to the `PgDn` key.

There is a compile time option that makes this command take an argument in lines instead of screenfuls. Look in `def.h` for the gory details.

M-V,C-Z **back-page**

Move backwards by pages. If an argument is given, it specifies the number of pages to move. If no argument is given, 1 is assumed. A page is a group of lines about 20% smaller than a window. If possible, dot is kept where it is; otherwise it it moved to the middle of the new page. On PCs, this function is bound to the `PgUp` key.

There is a compile time option that makes this command take an argument in lines instead of screenfuls. Look in `def.h` for the gory details.

M-< **goto-bob**

Move to the beginning of the buffer. Any argument is ignored. Dot is set to the first character of the first line in the buffer. On PCs, this function is bound to the `Control-Home` key.

M-> **goto-eob**

Move to the end of the buffer. Any argument is ignored. Dot is set to the first character in the fake line immediately after the buffer. The window is set to display dot near the center of the screen. On PCs, this function is bound to the `Control-End` key.

C-X G **goto-line**

This command moves the dot to a specific line number. If an argument is provided, the dot is moved to that line number. If no argument is provided, the command prompts on the echo line for a line number. This is useful for fixing a source file using error messages produced by a compiler.

C-X = **display-position**

This command displays information about the current position of the dot. The information is displayed on the echo line, and includes the following:

- The octal and hex values of the character at the dot.
- The current line number, in decimal.
- The current screen row and column, in decimal.
- The approximate position of the dot in the buffer, measured as a percentage of the buffer size.
- The number of characters in the buffer.

6 Inserting

All characters between hexadecimal 20 and 7E (blank through tilde), and all characters between hexadecimal A0 and FE (right hand graphics) are self-inserting. They are inserted into the buffer at the current location of dot, and dot moves 1 character to the right.

The **Tab** (or **C-I**) key is also self-inserting. By default, MicroEMACS has fixed tab settings at columns 9, 17, 25, and so on, but you can change the tab width using the **set-tab-size** command.

Any self-inserting character can be given an argument. This argument is used as a repeat count, and the character is inserted that number of times. This is useful for creating lines of `*` characters of a specific length, and other pseudo-graphic things.

Self-inserting keys are all bound to the function **ins-self**. As with any key, these keys can be rebound to other functions, but this is not always advisable.

Word wrapping can be performed by binding a key to the **ins-self-with-wrap** function. This function is described in the **Paragraphs** section below.

C-M, Return **ins-nl**

The Return key works just like you would expect it to work; it inserts a newline character. Lines can be split by moving into the middle of the line, and inserting a newline.

On some terminals with slow displays, if dot is positioned at end of line, and the line after the current line is a blank line, then **C-M** does not insert a newline, but simply moves to the first position on the following line. This lets you create a block of blank space (perhaps using **C-O** and then type text into it. This "feature" can be enabled at compile-time by setting the "NLMOVE" definition to 1 in `def.h`.

[**unbound**] **ins-nl-and-indent**

This command is like a **C-M** with indentation. It inserts a new line, and then inserts enough tabs and spaces to duplicate the indentation of the previous line. This is useful for entering heavily-indented programs in structured languages like PASCAL or C.

[**unbound**] **borland-indent**

This command is like a **ins-nl-and-indent**, but also attempts to indent according to the coding standards in use at Borland in the 1990s. If the previous line starts with `{`, or an argument of four (i.e. a single `Control-U`) is specified, indent by four spaces. If an argument of 16 (i.e. two `Control-U`s) is specified, reduce indentation by four spaces. Otherwise retain the same indentation.

C-J **gnu-indent**

This command is like a **ins-nl-and-indent**, but also attempts to indent according to the GNU coding standards in use at Cygnus in the 1990s. If the previous line starts with `{`, `if`, `while`, `for`, `else`, `case, or an argument of four (i.e. a single`Control-U`) is specified, indent by two spaces. If an argument of 16 (i.e. two`Control-U's) is specified, reduce indentation by two spaces. Otherwise retain the same indentation.

[**unbound**] **vmware-indent**

This command is like a **ins-nl-and-indent**, but also attempts to indent according to the coding standards in use at VMware in the 2000s. If the previous line starts with `{`, or an argument of four (i.e. a single `Control-U`) is specified, indent by three spaces. If an argument of 16 (i.e. two `Control-U`s) is specified, reduce indentation by three spaces. Otherwise retain the same indentation.

[**unbound**] **ruby-indent**

This command is like a **ins-nl-and-indent**, but also attempts to indent according to commonly accepted Ruby conventions. If the previous line starts with `{` or one of the many block-start keywords, or an argument of four (i.e. a single `Control-U`) is specified, indent by two spaces. If an argument of 16 (i.e. two `Control-U`s) is specified, reduce indentation by two spaces. Otherwise retain the same indentation.

C-O **ins-nl-and-backup**

This command creates blank lines. To be precise, it inserts a newline by doing a **C-M**, and then backs up by doing a **C-B**. If dot is at the start of a line, this will leave dot sitting on the first character of a new blank line.

C-Q, C-X Q **quote**

Characters which are special to MicroEMACS can be inserted by using this command. The next character after the **C-Q** or **C-X Q** is stripped of any special meaning. It is simply inserted into the current buffer. Any argument specified on the **C-Q** or **C-X Q** command is used as the insert repeat count.

The **C-Q** form of the command is the easiest to use. However, some terminals demand that MicroEMACS perform XON/XOFF processing. If this is the case, the **C-Q** will be eaten by low level terminal support, and will not be usable as a command. The **C-X Q** form can always be used.

C-T **twiddle**

Exchange the two characters on either side of the dot. If the the dot is at the end of the line, twiddle the two characters before it. Does nothing if the dot is at the beginning of the line. This command is supposedly useful for correcting common letter transpositions while entering new text.

M-Tab **set-tab-size**

Set the tab size to the value of the argument, which must be a positive number greater than 1 (the default tab size is 8). This only affects the how MicroEMACS displays tabs on the screen; it does not affect how tabs are saved when a file is written to disk. To change how MicroEMACS handles tabs when saving a file, see the **set-save-tabs** command.

M-I **set-save-tabs**

By default, MicroEMACS preserves tabs when it writes a file to disk. If you pass a zero argument to this command, MicroEMACS will convert tabs to spaces when writing a file; the number of spaces is determined by the tab size (which you can set using **set-tab-size**). If you pass a non-zero argument to this command, MicroEMACS will revert back to the default behavior, which is to preserve tabs.

[unbound] **just-one-space**

If the dot is currently sitting over a tab or a space, all consecutive tabs and spaces to the left and right of the dot are deleted. Then a single space is inserted, and the dot is moved 1 character to the right.

7 Deleting and Killing

There are two general classes of commands that remove text from the buffer: delete commands and kill commands. Delete commands remove text from the buffer, and throw it away. Kill commands remove text from the buffer, but save the text in a special place called the *kill buffer* . Kill commands clear the kill buffer only if the previous command was not a kill command. Multiple kill commands executed sequentially append text to the kill buffer.

C-D **forw-del-char**

Delete characters to the right of dot. If an argument is specified that number of characters is deleted. If no argument is specified then 1 character is deleted. In addition, if an argument is specified, then the command kills the text instead of deleting it. It fails if there are not enough characters to delete between dot and the end of the buffer. On PCs, this function is also bound to `Del`.

Rubout,C-H,Backspace **back-del-char**

Delete characters to the left of dot. If an argument is specified then that number of characters is deleted. If no argument is specified then 1 character is deleted. In addition, if an argument is specified, the rubout command kills the text instead of deleting it. The command fails if there is not enough text between the start of the buffer and dot.

C-K **kill-line**

This is the basic killing command. If there is no argument it kills from dot to the end of the line, unless dot is at the end of line, when it kills the end of line. If a positive argument is specified, **C-K** kills that many lines, including the newlines. If an argument of 0 is specified, **C-K** kills from the start of the current line to dot. Finally, if a negative argument is specified, **C-K** kills backwards over abs(arg) newlines. This command fails if there aren't enough characters left in the buffer to be killed.

C-W **kill-region**

Kill all of the text enclosed in the region, and put it into the kill buffer. If an argument is specified, then delete the text instead of killing it; this is useful when memory fills up and MicroEMACS can't allocate space for the kill buffer. The value of the argument, if any, is ignored.

C-Y **yank**

Insert the text from the kill buffer into the current buffer at dot. If an argument is specified, it specifies the number of times the text is yanked. If no argument is specified, yank the text back

once. Dot is advanced over the inserted text, as if the text had been typed in normally. Always succeeds.

M-W **copy-region**

Put all of the text enclosed in the region into the kill buffer, without deleting it from the current buffer. This is similar to **C-W** followed by **C-Y**, except that the buffer is not flagged as having been changed.

C-X C-O **del-blank-lines**

Delete blank lines around dot. If dot is sitting on a blank line, this command deletes all the blank lines above and below the current line. If it is sitting on a non blank line then it deletes all of the blank lines after the line. Any argument is ignored.

8 Words

MicroEMACS has commands that manipulate words. In full EMACS the characters that make up words can be changed by the user. In MicroEMACS, they are fixed, and include the upper and lower case letters, the dollar sign $ and the underline _. This set of characters is intended more for editing programs, but the word commands still work reasonably when editing text.

M-B **back-word**

The backward word command moves dot backward by words, stopping on the first character of the word. If an argument is specified, it is the number of words over which to move. The default argument is 1. On PC-DOS and FlexOS this function is also bound to `C-Left`.

M-C **cap-word**

This command moves forward over a word, converting all characters in the word to lower case except the first one, which is converted to upper case. If an argument is supplied, it must be positive, and it specifies the number of words over which to move.

M-D **forw-del-word**

The delete word command moves dot forward by words, and kills any characters over which it moves. If an argument is specified, it is the number of words over which to move. The default argument is 1.

M-F **forw-word**

The forward word command moves dot forward by words, stopping on the first non-word character. If an argument is specified, it is the number of words over which to move. The default argument is 1. On PC-DOS and FlexOS this function is also bound to `C-Right`.

M-L **lower-word**

This command moves forward over a word, converting it to lower case. If an argument is supplied, it must be positive, and it specifies the number of words over which to move.

M-C-H **back-del-word**

The backward delete word command moves backward by words, stopping on the first character of the word, and killing as it moves. If an argument is present, it specifies the number of words over which to move. The default argument is 1.

M-Rubout **back-del-word**

This command is the same as **M-C-H**. It exists only to preserve the symmetry between the word commands and the character commands (both backspace and rubout are backward delete character).

M-U **upper-word**

This command moves forward over a word, converting it to upper case. If an argument is supplied, it must be positive, and it specifies the number of words over which to move.

9 Paragraphs

MicroEMACS has commands that deal with paragraphs. A paragraph is a sequences of lines separated by one or more of the following:

- Blank lines.
- Lines beginning with a space or tab character.
- Lines beginning with the "@" character (SCRIBE, FinalWord, and Sprint command lines).
- Lines beginning with the "." character (NROFF and PROFF command lines).

There are commands for "filling" a paragraph, moving dot to the start or end of a paragraph, setting the fill column, and performing automatic word wrap.

[**unbound**] **set-fill-column**

This command sets the current fill column to its argument (remember that the argument is entered as `Control-U` and a decimal number preceding the command). If no argument is present, the column of the current location of the cursor (the dot) is used instead. The fill column is used by the **fill-paragraph** and **ins-self-with-wrap** commands. The default fill column is 70.

M-[**back-paragraph**

This command moves the dot to the beginning of the current paragraph. If the dot is not in a paragraph when this command is entered, it is moved to the beginning of the preceding paragraph. If an argument is provided, the dot is moved by that many paragraphs.

If you are using MicroEMACS with a serial terminal, you may have to type `ESCAPE` followed by two `[` characters to invoke this command. The reason is that ESCAPE-[is the prefix produced by function keys on VT-100 compatible terminals.

M-] **forw-paragraph**

This command moves the dot to the end of the current paragraph (actually to first separator line after the paragraph). If the dot is not in a paragraph when this command is entered, it is moved to the end of the following paragraph. If an argument is provided, the dot is moved by that many paragraphs.

M-J **fill-paragraph**

This command "fills" the current paragraph. It inserts or deletes spaces and line breaks between words as needed to cause the text of each line to be filled out to (but no farther than) the current fill column. The text is thus filled, but not right-justified. The dot is then placed at the end of the last line of the paragraph.

M-C-W **kill-paragraph**

This command deletes the current paragraph. If an argument is provided, that many paragraphs are deleted.

[**unbound**] **ins-self-with-wrap**

This command inserts the key that invoked it into the buffer at the current location of dot, and dot moves 1 character to the right. Then, if dot has gone past the current fill column, a line break is inserted in front of the first word in the line that extends past the fill column. Thus the current line is "filled", and a new line is created that contains the words from the current line that wouldn't fit within the fill column.

Normally, this command is not bound to any key. If you bind this command to the space key, the effect will be similar to the word-wrapping mode in MINCE.

10 Case Conversion

In addition to the word mode case conversion commands, MicroEMACS has commands to modify the case of large blocks of text. These commands should be used with caution because they cause major damage to (potentially) large areas of the buffer.

C-X C-L **lower-region**

The lowercase region command converts all of the characters between dot and mark into lower case.

C-X C-U **upper-region**

The uppercase region command converts all of the characters between dot and mark into upper case.

11 Searching and Spelling

Search commands move though the buffer, in either the forward or the reverse direction, looking for text that matches a search pattern. Search commands prompt for the search pattern in the echo line. The search pattern used by the last search command is remembered, and displayed in the prompt. If you want to use this pattern again, just hit carriage return at the prompt.

In search strings, all characters stand for themselves, and all searches are normally case insensitive. The case insensitivity may be defeated with the "fold-case" command, described below. The newline characters at the ends of the lines are considered to have hexadecimal value 0A, and can be matched by a linefeed (**Control-J**) in the search string.

A carriage return can be searched for by preceding it with `Control-Q` in the search string. On PC-DOS, CP/M and FlexOS this will not match the carriage return in the CR-LF character pair that normally terminates a line of text. It will only match a bare carriage return that has no following line feed.

MicroEMACS supports regular expression searches using a subset of POSIX regular expressions:

- `.` (dot) matches any character
- character classes (square brackets with optional ^ negation operator)
- groups (parentheses)
- alternatives (`|`)
- `^` (start of line) and `$` (end of line)
- `?` (zero or one occurrence)
- `*` (zero or more occurrences)
- `+` (one or more occurrences)

C-S, M-S **forw-search**

Search forward, from the current location, toward the end of the buffer. If found, dot is positioned after the matched text. If the text is not found, dot does not move.

The **C-S** form of the command is not usable if the terminal being used required XON/XOFF support. In fact, if you use this format of the command on such a terminal, it will hang until you type **C-Q**.

The **M-S** form of this command is easily entered on the Z-29 keyboard by pressing the `F1` key.

C-R **back-search**

Search reverse, from the current location, toward the front of the buffer. If found, dot is positioned at the first character of the matched text. If the text is not found, dot does not move.

M-C-S **forw-regexp-search**

Similar to **forw-search**, except that the search string is a regular expression, and searches cannot cross line boundaries.

M-C-R **back-regexp-search**

Similar to **back-search**, except that the search string is a regular expression, and searches cannot cross line boundaries.

M-C-F **fold-case**

Enable or disable case folding in searches, depending on the argument to the command. If the argument is zero, case folding is disabled, and searches will be sensitive to case. If the argument is non-zero, case folding is enabled, and searches will NOT be sensitive to case.

M-P **search-paren**

Search for a match of the character at the dot. If the character is a parenthesis or bracketing character, move the dot to the matching parenthesis or bracket, taking into account nesting and C-style comments. A parenthesis or bracketing character is one of the following: (){}[]<>

Pad-1,Shift-5 **search-again**

Repeat the last forw-search command (**C-S, M-S**) or prev-search command (**C-R, M-R**), without prompting for a string. This command is bound to two keys on the numeric keypad of the Z-29 terminal: the `1` key, and the shifted `5` key. On PCs, this function is also bound to `F9`.

C-X S **forw-i-search**

Enters incremental search mode, with the initial search direction being forward. In incremental search mode the following keys are recognized.

- **C-N** finds the next occurence of the string (if it is first thing typed, reuse the previous string).
- **C-P** finds the previous occurence of the string (if it is the first thing typed, reuse the previous string).
- **C-S** (or **C-F**) switches the search direction to forward, and finds the next occurrence of the string.
- **C-R** (or **C-B**) switches the search direction to reverse, and finds the next occurrence of the string.
- **C-Q** (or **C-^**) quotes the next character (allows searching for C-N, etc.).
- **ESC** exits from incremental search mode.
- **C-G** restores dot to original location before incremental search mode was entered, then exits from incremental search mode.
- **DEL** undoes the effect of the last character typed in the search string.
- **C-U**, **C-X**, **C-J**, and all non-Control characters accumulate into search string.
- All other control characters exit from incremental search mode and are interpreted as normal commands.

C-X R **back-i-search**

Enters incremental search mode, with the initial search direction being reverse. Otherwise identical to **C-X S**.

M-Q,M-% **query-replace**\index{M-%}

Search and replace with query. This command prompts for a search string and a replace string, then searches forward for the search string. After each occurrence of the search string is found, the dot is placed after the string, and the user is prompted for action. Enter one of the following characters:

- **space** or **,** (comma) causes the string to be replaced, and the next occurrence is searched.
- **.** (period) causes the string to replaced, and quits the search.

- **n** causes the string to be skipped without being replaced, and the next occurrence is searched.
- **!** causes all subsequent occurrences of the string to be replaced without prompting.
- **Control-G** quits the search without any further replacements.

Normally this command adjusts the capitalization of the new string to match the old string when it performs a replacement. You can defeat this "feature" if you prefix this command with an argument (the argument value is ignored).

M-R **replace-string**

Prompt for a search string and a replacement string, then search forward for all occurrences of the search string, replacing each one with the replacement string. Do not prompt the user for confirmation at each replacement, as in the **query-replace** command.

Normally this command adjusts the capitalization of the new string to match the old string when it performs a replacement. You can defeat this "feature" if you prefix this command with an argument (the argument value is ignored).

M-? **reg-query-replace**

Similar to **query-replace**, except that the search string is a regular expression, and the replacement string can contain the following special characters:

- `&` stands for the entire matched string.
- `\n`, where `n` is a digit in the range 0-9, stands for the nth matched group (where groups are delineated by parentheses in the regular expression pattern).
- `\` followed by either `\` or `&` stands for that character itself, without the leading `\`.

M-/ **rep-replace**

Similar to **reg-query-replace**, except that the user is prompted to confirm each replacement, as in **query-replace**.

C-X I **spell-region**

This command uses `ispell` to spell-check the current region (the text between the mark and the dot). At each misspelled word, MicroEMACS prompts for an action:

- **q** or **C-G** aborts the spell checking.
- **Space** ignores the misspelled word
- **a** ignores the misspelled word and adds it to ispell's list of words to ignore in the future.
- **0** to **9** replaces the misspelled word with one of up to ten suggestions; the suggestions are shown in the prompt on the echo line.
- **r** prompts for a string to replace the word.

M-$ **spell-word**

Similar to **spell-region**, except that it checks only the word under the cursor.

12 Files

C-X C-F **set-file-name**

This command prompts in the echo line for a file name, which becomes the new associated file name for the current buffer.

C-X C-R **file-read**

This command prompts in the echo line for a file name, then it deletes all of the text in the current buffer and reads in the file. The associated file name is set to the name of the file just read. The number of lines read is displayed in the echo line.

C-X C-I **file-insert**

This command prompts in the echo line for a file name, then reads in the file, inserting its contents at the dot in the current buffer. The number of lines read is displayed in the echo line. The file name associated with the current buffer is not changed.

C-X C-S, M-T **file-save**

This command writes the contents of the current buffer to its associated file. The "changed" flag for the current buffer is reset. It is an error to use this command in a buffer which lacks an associated file name. This command is a no-operation if the buffer has not been changed since the last write.

If you used the **-b** option when you invoked MicroEMACS, and this is first time a **file-save** command has been performed on the file, MicroEMACS will create a backup of the file. See the section **Starting** for more information on backups.

The **C-X C-S** form of the command is not usable if the terminal being used required XON/XOFF support. In fact, if you use this format of the command on such a terminal, it will hang until you type **C-Q**.

The **M-T** form of this command is easily entered on the Z-29 keyboard by pressing the `F2` key. On PCs, this function is also bound to `F2`.

C-X C-V **file-visit**

This command selects a file for editing. It prompts for a file name in the echo line. It then looks through all of the buffers for a buffer whose associated file name is the same as the file being selected. If a buffer is found, it just switches to that buffer. Otherwise it creates a new buffer, (fabricating a name from the last part of the new file name), reads the file into it, and switches to the buffer.

If the desired new buffer name is not unique (perhaps you tried to visit a file in some other directory with the same name as a file already read in) the command will prompt for a new buffer name. You can either supply a buffer name, or just type newline to overwrite the old buffer.

On PCs, this function is also bound to `F3`.

C-X C-W **file-write**

This command prompts in the echo line for a file name, then it writes the contents of the current buffer to that file. The "changed" flag for the current buffer is reset, and the supplied file name becomes the associated file name for the current buffer.

13 Keyboard Macros

Keyboard macros simplify a large class of repetitious editing tasks. The basic idea is simple. A set of keystrokes can be collected into a group, and then the group may be replayed any number of times.

There is only one keyboard macro. However, you can effectively have more than one macro by giving the current keyboard macro a name. Otherwise, when you define a new keyboard macro, the old one is erased.

You can also save macros for use in future editing sessions, with the ins-macro function.

C-X (**start-macro**

This command starts the collection of a keyboard macro. All keystrokes up to the next **C-X)** will be gathered up, and may be replayed by the execute keyboard macro command.

C-X) **end-macro**

This command stops the collection of a keyboard macro.

C-X E **execute-macro**

The execute keyboard macro command replays the current keyboard macro. If an argument is present, it specifies the number of times the macro should be executed. If no argument is present, it runs the macro once. Execution of the macro stops if an error occurs (such as a failed search).

[unbound] **name-macro**

This command saves the current keyboard macro and gives it a name. MicroEMACS prompts you for the name to assign to the macro. The macro name can then be used as an extended command (using `ESC X`), or can be bound to a key using **bind-to-key**, just as a normal command.

[unbound] **ins-macro**

This commands inserts the contents of a macro into the current buffer in a format that can be used later in a profile. MicroEMACS prompts for the name of the macro. If you don't enter a name, the current macro is used.

14 Buffer Management

Previous sections have made references to the text in "the buffer", which implied that there is only one buffer. This is not true; MicroEMACS allows any number of buffers, memory space permitting.

Each buffer has its own buffer name (a 16 character string), and optional associated file name, and a block of text. A value of dot and mark is also associated with any buffer that is not currently being displayed. This remembered value of dot and mark makes a buffer come back in approximately the same state as it was when it was hidden.

Also associated with each buffer is a changed flag. This flag is set when the text in the buffer is modified, and reset when the text in the buffer is written out to its associated file. MicroEMACS will always ask for confirmation before executing a command that would cause changed text to be lost.

C-X C-B **display-buffers**

Create a pop-up window on the screen, and display it in the name, size (in characters), associated file name, and changed flag of all buffers. This command works by creating a special buffer which contains the text of the display, and then selecting it in a window. You can switch into this window if you like. You can even edit the text. MicroEMACS makes no attempt to keep a buffer list which is on the screen updated as other buffers are edited; however, another **C-X C-B** command will cause the display to be updated in place. On PCs, this function is also bound to `F6`.

C-X B **use-buffer**

This command prompts for a buffer name, and then switches the buffer being displayed in the current window to that buffer. The buffer will be created if it does not exist.

If you do not enter a buffer name, this command will use the name of the last buffer that you switched from with **C-X B**. Thus, you can use **C-X B** repeatedly to switch between two buffers without entering their names each time.

[unbound] **forw-buffer**

This command switches the buffer being displayed to the next buffer in the buffer list. If the end of the buffer list is reached, switch to the first buffer in the list. The list of buffers can be displayed with **C-X C-B**. When this command is bound to a key, it is useful for quickly flipping among the files being edited. On PCs this command is bound to `F8`.

[unbound] **back-buffer**

This command is similar to **forw-buffer**, except that it switches the buffer being displayed to the previous buffer in the buffer list.

C-X K **kill-buffer**

This command prompts for a buffer name, and then destroys the buffer with that name. It will ask for permission to destroy the buffer if the text has been changed since it was written to the associated file. You cannot delete a buffer that is being displayed.

C-X C-Q **toggle-readonly**

This command toggles the read-only flag on the current buffer: if the buffer is currently read-only, it is made read-write; otherwise it is made read-only. This can be useful to counteract the effect of starting MicroEMACS with the `-r` option.

15 Window Management

MicroEMACS lets you have multiple windows on the screen. Each window has its own mode line, its own value of dot and mark, and its own associated buffer. If you have a buffer displayed in more that one window and you edit a line, it is updated in all windows.

A window is as only wide as can fit on your terminal. If a text line is too long to be displayed in a window, the last column is displayed as a `$` character. However, MicroEMACS will let you view long lines by automatically scrolling the window left or right if you move the cursor past the left or right edges of the window. When MicroEMACS performs a horizontal scroll, it will attempt to reframe the window so that the cursor is centered horizontally.

C-L **refresh**

This command clears the screen, and completely redisplays all of the text is all of the windows. It is useful if a line error has garbaged your screen.

If you give **C-L** an argument, it will attempt to split the screen vertically into that number of side-by-side pages. This has the effect of giving you a screen that has more rows but fewer columns than normal. To restore the screen to normal, give an argument of 1. Currently this feature only works with the ncurses display code; see **Building with GCC**.

C-X 2 **split-window**

This is the basic window making command. The current window is split in two. Each window is displaying the same buffer, and has the same value of dot and mark. The window must be at least three lines high, otherwise there isn't enough room for two text lines and the new mode line. Any argument is ignored.

After the you create a window, you usually switch it to a private buffer using **C-X B** or (perhaps more often) **C-X C-V**. Note that because both windows are displaying the same buffer, reading a file in one window with **C-X C-R** probably does not do what you want.

C-X 1 **only-window**

This is the basic window destroying command. All windows but the current window are removed from the screen. The current window grows to fill the vacated screen space. Any argument is ignored. On PCs, this command is also bound to `F10`.

C-X N, C-X O **forw-window**

Move to the next window down the screen. If the current window is the bottom window, move to the top window. Ignores any argument.

C-X P **back-window**

Move to the previous window up the screen. If the current window is the top window, move to the bottom window. Ignores any argument.

C-X Z **enlarge-window**

The current window is enlarged, if possible. Any argument is used as a "number of lines by which to grow". The default argument is 1. Screen space is stolen from the window immediately below the current window. If the current window is the bottom window on the screen then space is stolen from the window immediately above the current window. You cannot steal all of the lines away from a window.

C-X C-N **down-window**

Scroll the current window down. Any argument is used as a "number of lines by which to scroll" count. The default argument is 1. On PCs, this command is also bound to `C-PgDn`.

C-X C-P **up-window**

Scroll the current window up. Any argument is used as a "number of lines by which to scroll" count. The default argument is 1. On PCs, this command is also bound to `C-PgUp`.

C-X C-Z **shrink-window**

The current window is shrunk, if possible. Any argument is used as a "number of lines by which to shrink". The default argument is 1. Screen space is given to the window immediately below the current window. If the current window is the bottom window on the screen, the space is given to the window immediately above the current window. You cannot shrink a window out of existence.

M-! **reposition-window**

This command is used to control the line of a window upon which dot is displayed. If its argument is positive, then that number is taken to be the origin 1 line number of the current window upon which the line containing dot should be placed. If its argument is 0, then dot is moved to the center of the window. If the number is less then zero then it is taken to be the negation of the origin 1 line number of the current window starting at the bottom upon which the line containing dot should be placed. If no argument is supplied, a default argument of 1 is used; This lets **M-!** function as a "move dot to the top of the window" command, which is very useful.

C-X + **balance-windows**

This command adjusts the windows so that they all have approximately the same height. This is useful after several **split-window** commands have created some windows that are too small.

16 Messages

MicroEMACS has a message system that allows commands to display error or status messages that are too long to fit on the echo line. At present, only the **display-version** command uses the message system, but other commands may use it in the future.

[unbound] **display-message**

Display the message lines one at a time on the echo line, then enter a special message display mode, in which certain keys have the following meanings:

- **C-N, Space, Return**: go forward one line, and quit if the end of the message has been reached.
- **C-P, C-H, Backspace**: go backward one line.
- **C-G**: quit, leave the message unchanged.
- **C-C**: quit, erase the portion of the message already read, but leave the remainder of the message unchanged.

M-C-V **display-version**

Copy the version strings into the message buffer, then execute a **display-message** command (see above). Use this command to find out which version of MicroEMACS you are running.

17 Key Binding Commands

There are several commands in MicroEMACS that can be used to examine or change key bindings. These commands are not normally bound to a particular key. As with any unbound commands (also called "extended" commands), they can be used by entering **M-X** (`ESC X`), then typing the command name in response to the prompt `:` on the echo line.

[unbound] **display-bindings**

This command creates a pop-up window that contains a table of the current keyboard bindings. Each entry in the table contains the key name, and the name of the command that the key invokes.

You can save this table to a file by using **C-X N** to switch to the window displaying the table, then using **C-X C-W** to write the table to a file.

[unbound] **bind-to-key**

This command prompts the user with `Function:` for a command name, then prompts with `Key:` for a key (or key combination). The command is then bound to that key, so that all subsequent uses of that key cause the associated command to be executed. The command name can be any of those listed by the **display-bindings** command, or any of the commands described in this section.

[unbound] **help**

This command waits for the user to enter a key, then displays on the echo line the command name that is bound to that key. On PCs, this function is bound to `F1`.

18 Tags

MicroEMACS can use the source-code tagging programs `cscope` and `ctags` to make it easier to find functions and variables in C programs. The `find-` functions prompt for a name to find, but they also attempt to extract a name from the current buffer and location, to be used if you do not enter a name and simply hit `Enter`.

M-, **find-cscope**

This command prompts for an identifier, then uses `cscope` to find the first occurence of the identifier. Usually, the first occurrence is the one that defines the identifier. The command then visits the corresponding file and places the dot at the line containing the identifier. On PCs, this function is also bound to `F11`.

M-G **find-grep**

This command prompts for a string, then uses `cscope`'s "grep" function to find the first occurence of the string. The command then visits the corresponding file and places the dot at the line containing the string.

[unbound] **next-cscope**

This command uses `cscope` finds the next occurrence of an identifier that was found in a previous `find-cscope` or `find-grep` command. On PCs, this function is bound to `F12`.

M-. **find-tag**

This command prompts for an identifier, then reads the `TAGS` file (generated by `ctags`) to find an occurrence of the identifier. If no argument is present, the command finds the first occurrence; if an argument is present, the command finds the next occurence. The command then visits the corresponding file and places the dot at the line containing the identifier.

19 GCC errors

MicroEMACS is able to parse a file containing gcc error messages, and load the source files mentioned in the error messages. Typically, you would save the error messages to a file when running `make`:

```
make >&errs
```

Then load the resulting `errs` file into MicroEMACS and use the following command to process the error messages contained within:

M-C-E **gcc-error**

This command looks for the next gcc error message in the current buffer. If one is found, then a new window is opened if there is currently only one window; otherwise the next window below the current one is used. MicroEMACS reads into that window the file indicated in the error message, positions the cursor at the line and column given in the error message, and displays the remainder of the error message in the echo line. To find the next error location, switch back to the error buffer

(using the `back-window` command (**C-X P**) or the `forw-window` command (**C-X N**)), and issue this command again.

20 Undo

MicroEMACS supports an undo/redo facility. It saves information about the last 100 commands that modified a given buffer, so that these commands can be undone. Each buffer has its own set of undo records. MicroEMACS treats consecutive typing of normal (non-command) keys as a single operation; this makes it less tedious to undo large amounts of typing.

The most recent undo operation(s) can be reversed with the `redo` command. By using `undo` and `redo` in succession, you can go backwards and forwards in time through the recent history of a buffer. But if you modify the buffer in any way other than through `undo` or `redo`, the ability to `redo` will be lost until another `undo` is performed. This prevents conflicting changes from being made to a buffer.

C-X U **undo**

This command undoes the most recent operation that modified the current buffer. On PCs, this function is also bound to `F5`.

[**unbound**] **redo**

This command undoes the most recent undo. On PCs, this function is bound to `F7`.

21 Profiles

A profile is a file that contain a sequence of characters that MicroEMACS can read as if the file were a keyboard. Thus a profile is similar to a macro, but because it is a file it doesn't go away when you leave MicroEMACS. You can think of a profile as an editor command file.

When MicroEMACS starts, it reads and executes a startup profile, after it reads the file(s) you specify on the command line (if any). You might use the startup profile to set up some favorite key bindings, or define a macro, or read in more files.

You can use the **-p profile** option when you invoke MicroEMACS to specify the name of the startup profile. If you don't use this option, MicroEMACS will use a default profile name. The name of the default profile is `PE.PRO` on all systems except UNIX, where it is named `.pepro`. MicroEMACS will look for this profile in the current directory. If the profile is not found, MicroEMACS will look in a second directory, as follows:

- On UNIX and PC-DOS, the directory indicated by the environment string **HOME**, if defined.
- On FlexOS, the **home:** directory or device.
- On CP/M-68K, user area 0 of the current drive.
- On VMS, the **sys$login** directory.

A profile consists of a series of tokens separated by white space or newlines. A token can be a literal string, key name, or command name.

1. **Literal string:** a series of characters surrounded by double quotes ("). The characters within the quotes are interpreted by MicroEMACS exactly as if you had typed them on the keyboard. Certain control characters may be placed in the string using the following escape sequences:

\n	linefeed
\t	tab
\b	backspace
\r	carriage return
\f	form feed
\\	backslash

\" double quote

Other control characters can be placed in the string by preceding them with the Control-Q (quote) character.

A quoted string must always follow a command that normally prompt the user to enter a response string (for example, the search command **C-S**). The last character in the response string must be a carriage return (written as \r).

2. **Key name:** the name of a MicroEMACS key surrounded by brackets []. Key names use the conventions established in this manual. Examples:

   [C-F] means "Control-F"

   [M-C-V] means "ESC Control-F"

   [M-L] means "ESC L"

   [C-X E] means "Control-X E"

   [C-X C-S] means "Control-X Control-S"

   You can use the **display-bindings** extended command to get a partial list of key names, or see the **Wall Chart** section below.

   MicroEMACS converts the key name to the corresponding internal key code. Only one key name is allowed within a single pair of brackets.

3. **Commands name:** simply the name of any MicroEMACS command, whether it is bound to a key or not. MicroEMACS prefixes the command name with **ESC X**, and follows it with a **Return**, before interpreting the command. This simulates the keystrokes that you would enter at the keyboard to invoke an extended command.

4. **Decimal number:** a series of digits, optionally preceded by a minus sign ("-"). It is equivalent to typing Control-U, followed by the number, on the keyboard. Placing a number before a command is a convenient method of supplying a numeric argument to the command.

As an example, consider the following line from a profile:

```
bind-to-key "help\r" [m-h]
```

This is equivalent the following key sequence typed at the keyboard:

```
ESC X bind-to-key RETURN help RETURN ESC H
```

Note especially the '\r' character in the quoted string: this must be present because MicroEMACS always expects you to type **Return** in response to the `Function:` prompt of the **bind-to-key** command. This is true for all commands that prompt on the echo line for a reply.

Here is a profile that moves the cursor down by 10 lines, and which demonstrates the use of a numeric argument to a command:

```
10 forw-line
```

Here is a profile that changes all occurrences of the string "plugh" to "xyzzy" in the current file, then saves the changes and quits.

```
replace-string\index{replace-string} "plugh\r" "xyzzy\r" file-save quit
```

Here is a profile that causes the `Control-J` key to indent according to Ruby conventions.

```
bind-to-key "ruby-indent\r" [C-J]
```

You can automate this kind of global change with the **-p** option when you invoke MicroEMACS. If you name the above profile `junk.pro`, you can perform a global change on a file, without entering any MicroEMACS commands, by invoking MicroEMACS with the following:

```
pe -p junk.pro filename
```

You can tell MicroEMACS to read a profile at any time, with the following command.

C-X F **read-profile**

This command prompts you for the name of a profile. MicroEMACS then reads its subsequent commands from the specified profile, until the end of the file is encountered. While a profile is being processed, command errors are reported, but otherwise no screen activity takes place (unless the "echo" command is used in the profile). You cannot nest profiles by putting a [C-X F] or read-profile command in a profile. Execution of a profile does *not* stop if an error occurs (such as a failed search).

You can display messages on the echo line during profile processing (or at any other time) with the following command.

C-X C-E **echo**

This command prompts you to enter a line of text. MicroEMACS then displays the text on the echo line. This command might be useful for displaying progress during a time-consuming profile. Here is an example of the use of the echo command in a profile:

```
echo "Changing all CDOS references...\r"
replace-string "CDOS\r" "FlexOS\r"
```

Note the required carriage return (\r) at the end of the string.

22 Ruby Extensions

It is possible to extend MicroEMACS by writing commands in Ruby. To add Ruby support, you must specify the `--with-ruby` flag to `configure` when you build MicroEMACS. For example:

```
mkdir objruby
cd objruby
../configure --with-ruby
make
sudo make install # optional step
```

The resulting MicroEMACS is not linked directly with the Ruby runtime library. Instead, it loads the Ruby library dynamically as needed. This allows you to copy the MicroEMACS executable (`pe`) to a system where Ruby is not available, and it will still run, but without Ruby support.

A Ruby-enabled MicroEMACS will work only on a system that has the same version of Ruby as the build system. You will need to rebuild it for each system that has a different version of Ruby.

22.1 Initialization

In order for Ruby commands to run correctly, you will need to copy a helper file called `pe.rb`, located in the `ruby` subdirectory of the source code, to `/etc`. Running `sudo make install` after building a Ruby-enabled MicroEMACS will perform the copy. MicroEMACS will attempt to load the helper file when it starts. If it cannot load the file, you will not be able to use the Ruby extensions.

At startup, MicroEMACS will also attempt to load an initialization script that you have written. It will first look for a file called `.pe.rb` (note the leading dot) in your home directory, and load that if found. If it can't find `.pe.rb` in your home directory, it will look in your current directory. This will allow you to define your own custom commands on a per-user or per-project basis, as appropriate. The `ruby` subdirectory of the source code has examples that you can use or modify.

22.2 An example

Before delving into details about how to write commands in Ruby, let's look at an example. Here is a file called `gccerr.rb` that implements a command to parse gcc compiler errors and go to the relevant lines of code. This is essentially a rewrite of the built-in **gcc-error** command:

```
def gccerr(n)
  keepgoing = true
  while keepgoing
```

```
    l = $line
    if (l !~ /^In file included/ && l =~ /(.*):(\d+):(\d+): (.*)/)
      file = $1
      lno = $2
      col = $3
      err = $4
      if File.exist? file
        forw_line
        only_window
        split_window
        forw_window
        file_visit file
        goto_line lno.to_i
        forw_char col.to_i - 1
        echo "#{err}"
        return ETRUE
      else
        echo "File #{file} does not exist"
        return EFALSE
      end
    end
    keepgoing = forw_line == ETRUE
  end
  echo "No more gcc errors"
  return EFALSE
end

ruby_command "gccerr"
bind "gccerr", metactrl('e')
```

Some things to note about this example:

- It defines a new command called **gccerr**.
- Like all MicroEMACS commands written in Ruby, it takes a single parameter, which is the optional numeric argument.
- The new command invokes several built-in MicroEMACS functions, passing numeric parameters in some cases.
- It invokes built-in MicroEMACS commands, using their names with dashes replaced by underscores. For example, it invokes the `file-visit` command by calling the `file_visit` method.
- It uses the global variable `$line` to get the contents of the current line.
- It checks the status of the **forw-line** command by comparing it with the constant `ETRUE`, which corresponds to the constant `TRUE` in the C source code of MicroEMACS.
- After the definition of the command, there is code to tell MicroEMACS about the new command, and to bind it to the `M-C-E` key.
- If you want MicroEMACS to load this script automatically when it starts, rename it to `.pe.rb` and copy it either to your current directory or your home directory.

22.3 Ruby-related commands

MicroEMACS has several built-in commands related to Ruby extensions:

F6 **ruby-string**

This command prompts you to enter a line of Ruby code. MicroEMACS then passes the line to the Ruby interpreter. One common use of this command is to load a file containing Ruby code for a new command. For example, to load the code for the `gccerr` command described above, you could enter this command to `ruby-string`:

```
load 'PATH/gccerr.rb'
```

where you would replace PATH with the actual directory containing `gccerr.rb`.

[**unbound**] **ruby-command**

This command prompts you to enter the name of a Ruby function that implements a new command. MicroEMACS then enters the command into its symbol table but does not bind it to a key; you can use the `bind-to-key` command for that. The `gccerr` example above shows a use of this command.

[**unbound**] **ruby-load**

This command prompts the user for the name of a Ruby script, then loads that script. This is shortcut that has the same effect as using **ruby-string** and a `load` Ruby statement.

22.4 Calling Built-in Commands from Ruby

Ruby code can call built-in MicroEMACS commands (written in C) by invoking them as normal functions, but with the '-' characters in the names replaced by '_'. For example, invoke the **forw-char** function by calling `forw_char`.

You can pass an optional numeric parameter to a built-function. For example, to move the dot forward by 8 characters, use this code:

```
forw_char 8
```

Some commands prompt the user for one or more strings. You can supply these strings to a command by passing them as parameters. For example, to replace all occurrences of `Windows` to `Linux` in the current buffer, use this code:

```
replace_string "Windows", "Linux"
```

Some built-in commands prompt the user for a keystroke. Two examples are **help** and **bind-to-key**. These commands will not work as expected when invoked from Ruby, because as of this writing there is not a way to pass keycodes as additional parameters to commands.

Microemacs provides a `bind` helper function to work around the problem with the **bind-to-key** command. For example, the `gccerr.rb` code above used this helper to bind the **gccerr** command to the **M-C-E** key:

```
bind "gccerr", metactrl('e')
```

Commands return a trinary value indicating success, failure, or abort. In Ruby, these values are:

`ETRUE` The command succeeded.

`EFALSE` The command failed. For example, `forw_line` returns `EFALSE` if the dot is already at the last line, as we can see in the `gccerr` example above.

`EABORT` The command was aborted by Control-G.

The **echo** command is useful when debugging Ruby code. It displays a string on the echo line, so you can use it to display debug messages. For example, this code displays the current line number:

```
echo "line number is #{$lineno}"
```

22.5 Defining Commands in Ruby

You can create a new command in Ruby by first defining a function that takes a single numeric parameter. This parameter gives the numeric argument that the user typed as a prefix (using **C-U**). If the user didn't specify a numeric argument, the parameter will be `nil`.

Then use the **ruby-command** built-in command to inform MicroEMACS of the new command.

Referring to the `gccerr.rb` example above, we can see that the code first defines a new command function:

```
def gccerr(n)
  .. ruby code ...
end
```

Then it tells MicroEMACS about the new command:

```
ruby_command "gccerr"
```

Finally, it binds the new command to the **M-C-E** key:

```
bind "gccerr", metactrl('e')
```

22.5.1 Helper Functions

MicroEMACS provides several helper functions for use in Ruby commands.

`insert(string)` This function inserts the value of the `string` parameter into the current buffer at the dot. The string may contain newline characters, which are treated as line breaks.

`setmode(name)` This function deletes the current buffer's mode, if any. It then creates a mode called `name`, with an empty key binding table, and attaches it to the buffer. See the **Modes** section below for more information about modes.

`bind(name, key, mode=false)` This function binds the command whose name is the string `name` to the keycode `key`. If the `mode` parameter is present, and is `true`, the binding is attached to the current buffer's mode, if any. Otherrwise, the binding is made global, i.e., available in all buffers. See below for the helper functions that provide keycodes.

`reply(string)` This function prompts the user on the echo line with the specified string, then reads an input line from the user. It returns the input line without a terminating newline, or nil if the user aborts the input using Control-G.

`getkey` This function waits for the user to enter a keystroke, then returns a **Key** object describing the keystroke. See the next section for a description of the **Key** object.

`popup(string)` This function creates a pop-up window, with the contents specified by the `string` parameter. The string may contain newline characters. This function is useful for displaying error messages in a temporary window.

22.5.2 Keycodes

MicroEMACS also provides several helpers for encoding keycodes. All built-in commands in MicroEMACS take a keycode parameter, which contains the key that invoked the command. You can specify the keycode by passing it as a parameter when calling the command. As of this writing, the only command that looks at the keycode is **ins-self**. Given that fact, the following example inserts an 'x' character in to the current buffer:

```
ins_self key('x')
```

The `bind` helper function, described above, also takes a keycode parameter.

Keycodes can be specified using one of the following helper functions. These helpers all take a single parameter, which is an ordinary ASCII character.

- `key`: specifies an ordinary, unmodified character. For example, `key('c')` means the character 'c'.
- `ctrl`: specifies a control character. For example, `ctrl('c')` means **C-C** (Control-C).
- `meta`: specifies a meta character. For example, `meta('c')` means **M-C** (Escape C or Alt-C).
- `ctlx`: specifies a character with the `C-X` prefix. For example, `ctlx('c')` means **M-X C** (Control-X C).
- `metactrl`: specifies a combination of `meta` and `ctrl`. For example, `metactrl('c')` means **M-C-C** (Escape Control-C).
- `ctlxctrl`: specifies a combination of `ctlx` and `ctrl`. For example, `ctlxctrl('c')` means **C-X C-C** (Control-X Control-C).

These helpers all return an object of the class **Key**. This object contains the raw keycode as used internally by MicroEMACS, and provides methods for examining the keycode. Here are the **Key** methods:

`ctrl?` Returns true if the key is a control key.

`meta?` Returns true if the key is a meta key (i.e., has an Escape prefix).

`ctlx?` Returns true if the key is a Control-X key (i.e., has a Control-X prefix).

`normal?` Returns true if the key is a "normal" key (i.e., is not a control, meta, or Control-X key).

`to_i` Returns the key's raw keycode.

`char` Returns the normal character portion of the keycode, without any control, meta, or Control-X flags. As an example, the `char` of the Control-G keycode is the character 'G'.

`to_s` Returns a human readable string for the keycode. As an example, the `to_s` of the Control-G keycode is 'C-G'.

22.5.3 Global variables

MicroEMACS provides several global virtual variables that may be both read and written in Ruby code.

`$line` This variable contains the current line (the line containing the dot), with a newline character appended if this is not the last line in the buffer. Writing to this variable causes the current line to be replaced with the specified string. A newline at the end of the string is removed, but newlines at other positions in the string are left unchanged and cause line breaks.

`$char` This variable contains the character at the dot. Writing to this variable replaces the character at the dot with the specified string (which can be of any length).

`$lineno` This variable contains the line number of the line containing the dot. The value is 1-based, for compatibility with the **goto-line** function. Writing to this variable causes the dot to be moved to the specified line.

`$offset` This variable contains the offset into the current line of the dot. The value is 0-based, so that it can be used as an index into `$line`. Writing to this variable moves the dot to the specified offset within the current line.

`$filename` This variable contains the current buffer's filename. Writing to this variable changes the current buffer's filename.

`$tabsize` This variable contains the current tab width. Writing to this variable sets the tab width, as in the **set-tab-size** command.

`$fillcol` This variable contains the current fill column for paragraph justification. Writing to this variable sets the fill column, as in the **set-fill-column** command.

`$bflag` This variable contains the current buffer's flags, and can be read or written. The flags are an OR of these values: BFCHG (buffer has changed), BFBAK (buffer needs a backup), and BFRO (buffer is read-only). For example, this code:

```
$bflag &= ~BFCHG
```

turns off the "buffer changed" flag. This is a dangerous operation, because it could result in data loss.

22.6 Exceptions

If an exception occurs in Ruby code, MicroEMACS will open a temporary window containing the exception information, including a backtrace.

In the unlikely event that the Ruby interpreter crashes with a segfault, it prints complete exception information to the terminal, but the output is difficult to read because MicroEMACS puts the terminal into "raw" mode. If you need to see the exception information, you can restart MicroEMACS with stderr redirected to a file:

```
pe 2>ruby.log
```

Then, if you can reproduce the crash, the file `ruby.log` will contain the exception information.

22.7 Aborting Ruby Commands

If your Ruby code is taking too long to run, and you want to stop it, you will need to send it a signal from another terminal window. In that window, find the ID of the process that is running MicroEMACS, using a command such as this:

```
ps -x | grep pe
```

Then using the process ID that this command displays, kill the Ruby code using:

```
kill -SIGINT <id>
```

The helper code in `pe.rb` catches this signal and raises an exception that aborts the errant Ruby code and return control to MicroEMACS.

23 Modes

If MicroEMACS has been built with Ruby support, it will also support the notion of modes, which are similar to major modes in Emacs. A mode consists of a name (which is arbitrary) and a set of key bindings local to that mode, and is attached to a specific buffer. By default, buffers in MicroEMACS do not have modes, but you can provide support for modes by loading Ruby support for them. Modes can be used to implement features for particular types of code, or to provide special types of buffers that are not treated solely as plain text.

23.1 Mode initialization

When MicroEMACS reads a file into a buffer, or visits a file (which may or may not exist yet), it calls the Ruby function `initmode`, which lives in the file `pe.rb`, and which gets loaded whenever MicroEMACS starts. This function attempts to determine the name of the mode associated with the file being edited. First, it examines the first few lines in the file itself for a line containing a string in the following format:

```
-*-MODE-*-
```

where `MODE` is the name of the mode associated with the file.

If such a string cannot be found, `initmode` then examines the name of the file itself, and attempts to find a match in the `$modetable` array in `pe.rb`. This table associates filename patterns with mode names. The table has very few entries by default, but you can expand it as necessary by editing `pe.rb` directly, or by overwriting it or adding to it in your own Ruby extension.

If `initmode` can determine the mode name, it calls the function `MODE_mode`, where `MODE` is the name of the mode. For example, if `initmode` determines that the mode name is `c`, it calls the function `c_mode`, if it exists. This function is called the mode hook, and it is responsible for performing all necessary initialization for the mode. If the mode hook does not exist, MicroEMACS does not report an error.

23.2 Simple mode example

To see how a rudimentary mode is written, look at the file `mode.rb` in the MicroEMACS source code. (This file should also have been installed in `/usr/local/share/pe`, if you built MicroEMACS and ran `make install`.)

Here is the code for `c_mode`, the hook for C files, mentioned above:

```
def c_mode
  setmode "C"
  bind "gnu_indent", ctrl('j'), true
  echo "This is C mode"
end
```

The first line in this function creates a new mode for the current buffer, and assigns it the name “C”. The name is arbitrary, and is used only for displaying in the mode line for the buffer. But `setmode` must be called; otherwise a mode will not exist for the current buffer, and attempts to mode-local key bindings will not succeed.

The second line in this function binds the MicroEMACS function `gnu-indent` to the **C-J** key. This binding is local to this mode only. If you switch to a buffer that has a different mode, or no mode, the binding for **C-J** may well be different.

The third line in this function is present only for debugging purposes. It can be safely deleted.

You can load this rudimentary mode support automatically by adding the following line to `~/pe.rb` or `./pe.rb`:

```
load 'mode.rb'
```

23.3 A more complicated mode example

The file `dired.rb`, also provided with MicroEMACS, is a more elaborate example of a mode. It implements a directory browser similar to the dired mode in Emacs. Unlike the simple example shown above, it provides both global and mode-specific key bindings, and it does not provide a mode hook function. Instead, it provides a new command, `dired`, that is globally bound to the keystroke **C-X D**:

```
ruby_command "dired"
bind "dired", ctlx('d')
```

The file `dired.rb` also creates three new commands but does not bind them to keystrokes immediately:

```
ruby_command "visitfile"
ruby_command "openfile"
ruby_command "displayfile"
```

The initialization of the mode happens in the `dired` function when you enter the keystroke **C-X D**. This function prompts you for a directory name, then calls `showdir` to open a view on the directory.

The `showdir` function opens a new window with a buffer called `*dired*`, to avoid conflict with existing buffers. It clears any existing contents of the buffer. It then runs `/bin/ls -laF` to load the buffer with a directory listing. The first line in the buffer contains the directory name, and each subsequent line contains information about a file in that directory, as provided by `ls`. If the name of a file ends in a '/' character, that file is actually a subdirectory. Finally, `showdir` marks the buffer as read-only.

The `showdir` function then performs some string matching to determine that starting column for filenames in the directory listings. Finally, it creates a mode for the directory listing and attaches three key bindings to it:

```
setmode "dired"
bind "visitfile", ctrl('m'), true
bind "openfile", key('o'), true
bind "displayfile", ctrl('o'), true
```

The `bind` calls create key bindings for the three new commands that were defined earlier. These bindings perform three distinct actions on the file under the cursor (called the "selected" file):

- The `Enter` key opens the selected file in a new window, replacing the current dired window (which still exists).
- The `o` key splits the screen into two windows, one containing the dired buffer, and the other containing the selected file. It then moves the cursor to the selected file.
- The `C-O` key is similar to the `o` key, except that it does not move the cursor to the selected file.

You can load the dired mode support automatically by adding the following line to `~/pe.rb` or `./pe.rb`:

```
load 'dired.rb'
```

24 UTF-8 and Unicode

MicroEMACS supports reading and writing text files encoded with UTF-8, a byte-oriented encoding of Unicode. UTF-8 can be thought of as a superset of ASCII: bytes less then 0x80 are identical to ASCII, and bytes greater than or equal to 0x80 are always parts of UTF-8 sequences. UTF-8 characters vary in length from one byte (ASCII) to six bytes, though four bytes is the longest typically seen.

Internally, Microemacs stores lines of text in their original UTF-8 encoding, but displays multibyte UTF-8 sequences to the user as single Unicode characters in the range 0x80 to 0xffff. Unicode characters greater then 0xffff are not supported; in practice these characters are extremely rare.

For the most part, MicroEMACS is able to display UTF-8 characters correctly, with the exception of non-spacing or combining characters. The display of these characters is undefined: sometimes they show up as modifiers of subsequent characters, or as blanks, or as a lowercase 'x' with a modifier.

On Linux, most terminal programs support a standard method for entering Unicode characters at the keyboard: hold down Ctrl and Shift, then press and release 'u', then release Ctrl and Shift, then enter the hex digits of the Unicode character followed by the Enter key. This method works when entering characters at prompts in the echo line, or while entering text in the edit buffer.

If this method doesn't work, you can still enter Unicode characters in the edit buffer using the following command:

M-C-U **unicode**

This command prompts you to enter a line of text containing the hexadecimal values of one or more Unicode characters. The hex values must not have a '0x' prefix, or any other prefix, and must be separated by spaces. MicroEMACS will then insert the corresponding UTF-8 characters into the current buffer.

As an example, entering the string `e0 e1 e2` would insert the characters `àáâ` into the buffer.

25 Building a MicroEMACS

All versions of MicroEMACS are built from the two sets of source files. One set is independent of operating system and terminal, and the other set is dependent.

Compile time options for the independent modules are selected by setting compilation switches in `def.h`, and then letting conditional compilation do the right thing.

The dependent modules are briefly described below.

25.1 Operating System

MicroEMACS runs on several operating systems, including Linux, FreeBSD, and Windows. Support code for other operating systems has been lost (in the distant past, these included CP/M-86 and MS-DOS on the DEC Rainbow, VMS on the VAX, CP/M-68K, GEMDOS, and FlexOS V60/68K/286). The following modules contain code dependencies on the operating system:

- `ttyio.c` - low level terminal I/O; independent of terminal type.
- `spawn.c` - subjob creation.
- `fileio.c` - low level file handling.
- `bcopy.s` or `bcopy.asm` - fast byte copy and fill functions.

Adding a new operating system consists mostly of changing these files, and the header file `sysdef.h`.

25.2 Terminal Support

MicroEMACS supports several kinds of terminals: those supporting ncurses or termcap, and the native Windows text console (the code for real-mode PC displays and OS/2 terminal windows has been lost). The following modules contain code dependencies on the terminal type:

- `tty.c` - high-level terminal support.
- `ttyio.c` - low-level terminal support.
- `ttykbd.c` - keyboard dependencies and extensions.

Changing terminal type consists mostly of changing these files, and the header file `ttydef.h` These files are located in separate per-terminal subdirectories of the `tty` directory.

Some terminals have memory mapped displays, or interfaces that act as such. These include ncurses and Windows text consoles. Support for these displays is enabled by setting the MEMMAP switch in `ttydef.h` to 1. This eliminates the fancy Gosling screen update code in `display.c`, and enables writing directly to screen memory (or to a screen buffer that the terminal interface library later writes to the screen).

To support a new memory-mapped display, you must provide a `putline` function for writing lines to the display. On old DOS-base systems, this code was written in assembly language, but on modern terminals it is written in C and placed in `tty.c`.

25.3 Building with GCC

To build MicroEMACs on Linux, FreeBSD, or Windows using Cygwin or MinGW, use these commands:

```
mkdir obj
cd obj
../configure
make # gmake on FreeBSD
```

You can supply one or more optional parameters to the `configure` command:

`--with-termcap` Use this option to make MicroEMACS use the **terminfo** / **termcap** libraries for screen management, instead of the default **ncursesw** library. This option will not work on Windows.

`--enable-debug` Use this option to compile and build MicroEMACS with debugging information, so that it can be debugged with gdb.

`--with-ruby` Use this option to build support for Ruby extensions into MicroEMACS. This option will not work on Windows or FreeBSD. See the **Ruby Extensions** section above for more information.

26 Wall Chart

Here is a list of the current key bindings in MicroEMACS. The terminal-dependent key bindings are presented at the end of this section.

Insert and Delete

Return	ins-nl
C-J	gnu-indent
C-O	ins-nl-and-backup
C-T	twiddle
M-L	lower-word
M-U	upper-word
M-C	cap-word
M-D	forw-del-word
C-D	forw-del-char
Rubout	back-del-char
Backspace	back-del-char
M-Rubout	back-del-word
M-Backspace	back-del-word
C-K	kill-line
C-Y	yank
C-X C-O	del-blank-lines
M-C-U	unicode
M-Tab	set-tab-size
M-I	set-save-tabs

Little Moves

C-F	forw-char
C-B	back-char
M-F	forw-word
M-B	back-word
C-N	forw-line
C-P	back-line
C-A	goto-bol
C-E	goto-eol

Big Moves

C-V	forw-page
C-Z	back-page
M-V	back-page
M-<	goto-bob
M->	goto-eob
C-X G	goto-line

Paragraphs

M-[	back-paragraph
M-]	forw-paragraph
M-J	fill-paragraph
M-C-W	kill-paragraph

Region Commands

C-@,C-space	set-mark
M-.	set-mark
C-X C-X	swap-dot-and-mark
C-X C-L	lower-region
C-X C-U	upper-region
M-W	copy-region
C-W	kill-region

Search and Replace

C-S	forw-search
M-S	forw-search
C-R	back-search
M-C-F	fold-case
M-P	search-paren
M-Q	query-replace
M-%	query-replace
M-R	replace-string
C-X S	forw-i-search
C-X R	back-i-search
M-C-S	forw-regexp-search
M-C-R	back-regexp-search
M-/	reg-replace
M-?	reg-query-replace

File and System Operations

C-X C-I	file-insert
C-X C-V	file-visit
C-X C-R	file-read
C-X C-W	file-write
M-T	file-save
C-X C-S	file-save
C-X C-F	set-file-name
C-X B	use-buffer
C-X K	kill-buffer
C-L	refresh
C-Q	quote
C-X Q	quote
M-X	extended-command
C-C	spawn-cli
C-X C-G	abort
C-G	abort
M-C-G	abort
C-X C-C	quit
C-X C	jeff-exit
C-X I	spell-check

Windows

C-X 1	only-window
C-X 2	split-window
C-X +	balance-windows
M-!	reposition-window
C-X Z	enlarge-window
C-X C-Z	shrink-window
C-X N	forw-window
C-X O	forw-window
C-X P	back-window
C-X C-P	up-window
C-X C-N	down-window
M-C-R	display-message
M-C-V	display-version
C-X =	display-position
C-X C-B	display-buffers
C-X C-Q	toggle-readonly

Macros and Profiles

C-X (	start-macro
C-X)	end-macro
C-X E	execute-macro
C-X F	read-profile
C-X C-E	echo

Tags, GCC

M-,	find-cscope
M-G	find-grep
M-.	find-tag
M-C-E	gcc-error

PC Function Keys

F1	help
F2	file-save
F3	file-visit
F4	quit
F5	undo
F6	display-buffers
F7	redo
F8	forw-buffer
F9	search-again
F10	only-window
F11	find-cscope
F12	next-cscope
Up	back-line
Down	forw-line
Left	back-char
Right	forw-char
PgUp	back-page
PgDn	forw-page
Home	goto-bol
End	goto-eol
C-Left	back-word
C-Right	forw-word
C-PgDn	down-window
C-PgUp	up-window
C-Home	goto-bob
C-End	goto-eob
Insert	yank
Delete	forw-del-char

Zenith Z-19/29 Keypad

Pad-1	search-again
Pad-2	forw-line
Pad-4	back-char
Pad-5	forw-search
Pad-6	forw-char
Pad-7	forw-del-word
Pad-8	back-line
Pad-9	kill-line
Pad-Enter	ins-nl
Blue-0	ins-nl-and-backup
Blue-2	goto-eob
Blue-4	goto-bol
Blue-5	query-replace
Blue-6	goto-eol
Blue-7	yank
Blue-8	goto-bob
Blue-9	yank
Blue-Period	abort
Blue-Enter	ins-nl
M-A	back-page
M-B	forw-page
M-C	forw-word
M-D	back-word

Index